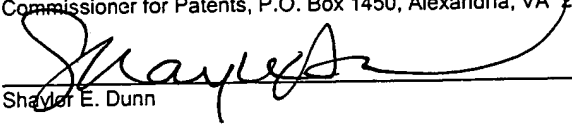


"Express Mail" mailing number: ED114862290US
Date of Deposit June 23, 2006
I hereby certify that this correspondence is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 C.F.R. 1.10 on the date indicated above and is addressed to Mail Stop RCE, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.


Shayer E. Dunn

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of: Injong Rhee

Group Art Unit: 2667

Serial No.: 09/989,957

Examiner: Grey, Christopher P.

Filed: November 21, 2001

Docket No. 297/123/2

Confirmation No.: 1668

For: METHODS AND SYSTEMS FOR RATE-BASED FLOW CONTROL BETWEEN A
SENDER AND A RECEIVER

SUPPLEMENTAL INFORMATION DISCLOSURE STATEMENT

Mail Stop RCE
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

In accordance with 37 C.F.R. 1.56, 1.97, and 1.98, applicants' undersigned attorney brings to the attention of the Patent and Trademark Office the documents listed on the attached Form PTO-1449. Copies of the references as well as Form PTO-1449 are attached hereto. This is not to be construed as a representation that a search has been made or that a reference is relevant merely because cited.

Early passage of the subject application to issue is earnestly solicited.

Serial No.: 09/989,957

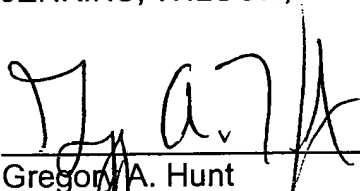
A check in the amount of \$395.00 is enclosed for the Request for Continued Examination fee. However, the Commissioner is authorized to charge any deficiencies of payment or credit any overpayments associated with the filing of this correspondence to Deposit Account No. 50-0426.

Respectfully submitted,

JENKINS, WILSON, TAYLOR & HUNT, P.A.

Date: June 23, 2006

By:



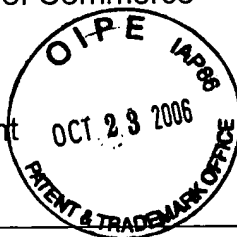
Gregory A. Hunt
Registration No. 41,085
Customer No. 25297

297/123/2 GAH/sed

Enclosures

FORM PTO-1449 U.S. Department of Commerce
Patent and Trademark Office

List of Documents Cited by Applicant



Application No.:	09/989,957
Filing Date:	November 21, 2001
First Named Inventor:	Injong Rhee
Group:	2667
Examiner:	Grey, Christopher P.
Attorney Docket No.:	297/123/2

U.S. PATENT DOCUMENTS

Examiner Initial	Cite No.	Document Number	Publication Date	Name of Patentee or Applicant of Cited Document	Pages, Columns, Lines, where relevant passages or relevant figures appear

FOREIGN PATENT DOCUMENTS

Examiner Initials	Cite No.	Document Number (country code, no., kind code (if known))	Publication Date	Name of Patentee or Applicant	Pages, columns, lines where relevant passages appear	T

OTHER DOCUMENTS

Examiner Initials	Cite No.	Include Author (in CAPITAL LETTERS), Title, Journal, Date, Pertinent Pages, Etc.	T
	1	Whetten et al., "Reliable Multicast Transport Building Blocks for One-to-Many Bulk-Data Transfer," IETF Internet-Draft, http://www.ietf.org/rfc/rfc3048.txt , pgs. 1-19 (January 2001).	
	2	Lee et al., "An Application Level Multicast Architecture for Multimedia Communications in the Internet," IRTF Reliable Multicast Research Group (November 1999).	
	3	Ramesh et al., "Issues in Model-Based Flow Control," IRTF Reliable Multicast Research Group, pgs. 1-14 (November 1999).	
	4	Luby et al., "Heterogeneous Multicast Congestion Control Based on Router Packet Filtering," IRTF Reliable Multicast Research Group, pgs. 1-13 (May 31, 1999).	
	5	Bhattacharyya et al., "The Loss Path Multiplicity Problem for Multicast Congestion Control," In Proceedings of IEEE INFOCOM, pgs. 856-863 (1999).	
	6	Padhye et al., "A Model Based TCP-Friendly Rate Control Protocol," In Proceedings of the Ninth International Workshop on Network and Operating Systems Support for Digital Audio and Video (1999).	

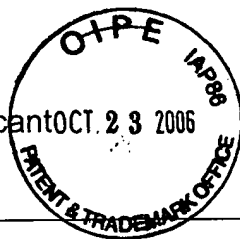
	7	Tuan et al., "Multiple Time Scale Redundancy Control for QoS-Sensitive Transport of Real-Time Traffic," In Proceedings of INFOCOM, pgs. 1683-1692 (2000).	
	8	Li et al., "HPF: A Transport Protocol for Supporting Heterogeneous Packet Flows in the Internet," Research Paper, Coordinated Sciences Laboratory, University of Illinois at Urbana-Champaign, pgs. 543-550 (1998).	
	9	Turletti et al., "Experiments with a Layered Transmission Scheme Over the Internet," Technical Report RR-3296, pgs. 1-26 (November 1997).	
	10	Mathis et al., "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm," ACM Computer Communication Review, 27(3), pgs. 67-82 (July 1997).	
	11	Floyd, "Connections with Multiple Congested Gateways in Packet-Switched Networks Part 2: Two-Way Traffic," Unpublished draft, pgs. 1-8 (December 1991).	

EXAMINER _____ DATE CONSIDERED _____

*Examiner Initial if reference considered, whether or not citation is in conformance with MPEP 609; draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant.

FORM PTO-1449 U.S. Department of Commerce
Patent and Trademark Office

List of Documents Cited by Applicant



Application No.:	09/989,957
Filing Date:	November 21, 2001
First Named Inventor:	Injong Rhee
Group:	2667
Examiner:	Grey, Christopher P.
Attorney Docket No.:	297/123/2

U.S. PATENT DOCUMENTS

Examiner Initial	Cite No.	Document Number	Publication Date	Name of Patentee or Applicant of Cited Document	Pages, Columns, Lines, where relevant passages or relevant figures appear

FOREIGN PATENT DOCUMENTS

Examiner Initials	Cite No.	Document Number (country code, no., kind code (if known))	Publication Date	Name of Patentee or Applicant	Pages, columns, lines where relevant passages appear	T

OTHER DOCUMENTS

Examiner Initials	Cite No.	Include Author (in CAPITAL LETTERS), Title, Journal, Date, Pertinent Pages, Etc.	T
	1	Whetten et al., "Reliable Multicast Transport Building Blocks for One-to-Many Bulk-Data Transfer," IETF Internet-Draft, http://www.ietf.org/rfc/rfc3048.txt , pgs. 1-19 (January 2001).	
	2	Lee et al., "An Application Level Multicast Architecture for Multimedia Communications in the Internet," IRTF Reliable Multicast Research Group (November 1999).	
	3	Ramesh et al., "Issues in Model-Based Flow Control," IRTF Reliable Multicast Research Group, pgs. 1-14 (November 1999).	
	4	Luby et al., "Heterogeneous Multicast Congestion Control Based on Router Packet Filtering," IRTF Reliable Multicast Research Group, pgs. 1-13 (May 31, 1999).	
	5	Bhattacharyya et al., "The Loss Path Multiplicity Problem for Multicast Congestion Control," In Proceedings of IEEE INFOCOM, pgs. 856-863 (1999).	
	6	Padhye et al., "A Model Based TCP-Friendly Rate Control Protocol," In Proceedings of the Ninth International Workshop on Network and Operating Systems Support for Digital Audio and Video (1999).	

	7	Tuan et al., "Multiple Time Scale Redundancy Control for QoS-Sensitive Transport of Real-Time Traffic," In Proceedings of INFOCOM, pgs. 1683-1692 (2000).	
	8	Li et al., "HPF: A Transport Protocol for Supporting Heterogeneous Packet Flows in the Internet," Research Paper, Coordinated Sciences Laboratory, University of Illinois at Urbana-Champaign, pgs. 543-550 (1998).	
	9	Turletti et al., "Experiments with a Layered Transmission Scheme Over the Internet," Technical Report RR-3296, pgs. 1-26 (November 1997).	
	10	Mathis et al., "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm," ACM Computer Communication Review, 27(3), pgs. 67-82 (July 1997).	
	11	Floyd, "Connections with Multiple Congested Gateways in Packet-Switched Networks Part 2: Two-Way Traffic," Unpublished draft, pgs. 1-8 (December 1991).	

EXAMINER _____ DATE CONSIDERED _____

*Examiner Initial if reference considered, whether or not citation is in conformance with MPEP 609; draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant.



Network Working Group
Request for Comments: 3048
Category: Informational

B. Whetten
Talarian
L. Vicisano
Cisco
R. Kermode
Motorola
M. Handley
ACIRI 9
S. Floyd
ACIRI
M. Luby
Digital Fountain
January 2001

Reliable Multicast Transport Building Blocks for One-to-Many
Bulk-Data Transfer

Status of this Memo

This memo provides information for the Internet community. It does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Copyright Notice

Copyright (C) The Internet Society (2001). All Rights Reserved.

Abstract

This document describes a framework for the standardization of bulk-data reliable multicast transport. It builds upon the experience gained during the deployment of several classes of contemporary reliable multicast transport, and attempts to pull out the commonalities between these classes of protocols into a number of building blocks. To that end, this document recommends that certain components that are common to multiple protocol classes be standardized as "building blocks". The remaining parts of the protocols, consisting of highly protocol specific, tightly intertwined functions, shall be designated as "protocol cores". Thus, each protocol can then be constructed by merging a "protocol core" with a number of "building blocks" which can be re-used across multiple protocols.

Whetten, et al.

Informational

[Page 1]

□

RFC 3048

RMT Building Blocks

January 2001

Table of Contents

1 Introduction	2
1.1 Protocol Families	5
2 Building Blocks Rationale	6
2.1 Building Blocks Advantages	6
2.2 Building Block Risks	7
2.3 Building Block Requirements	8
3 Protocol Components	8
3.1 Sub-Components Definition	9
4 Building Block Recommendations	12
4.1 NACK-based Reliability	13
4.2 FEC coding	13
4.3 Congestion Control	13
4.4 Generic Router Support	14
4.5 Tree Configuration	14
4.6 Data Security	15
4.7 Common Headers	15
4.8 Protocol Cores	15
5 Security	15
6 IANA Considerations	15
7 Conclusions	16
8 Acknowledgements	16
9 References	16
10 Authors' Addresses	19
11 Full Copyright Statement	20

1. Introduction

RFC 2357 lays out the requirements for reliable multicast protocols that are to be considered for standardization by the IETF. They include:

- o Congestion Control. The protocol must be safe to deploy in the widespread Internet. Specifically, it must adhere to three mandates: a) it must achieve good throughput (i.e., it must not consistently overload links with excess data or repair traffic), b) it must achieve good link utilization, and c) it must not starve competing flows.
- o Scalability. The protocol should be able to work under a variety of conditions that include multiple network topologies, link speeds, and the receiver set size. It is more important to have a good understanding of how and when a protocol breaks than when it works.

Whetten, et al.

Informational

[Page 2]

□

RFC 3048

RMT Building Blocks

January 2001

- o Security. The protocol must be analyzed to show what is necessary to allow it to cope with security and privacy issues. This includes understanding the role of the protocol in data confidentiality and sender authentication, as well as how the

protocol will provide defenses against denial of service attacks.

These requirements are primarily directed towards making sure that any standards will be safe for widespread Internet deployment. The advancing maturity of current work on reliable multicast congestion control (RMCC) [HFW99] in the IRTF Reliable Multicast Research Group (RMRG) has been one of the events that has allowed the IETF to charter the RMT working group. RMCC only addresses a subset of the design space for reliable multicast. Fortunately, the requirements it addresses are also the most pressing application and market requirements.

A protocol's ability to meet the requirements of congestion control, scalability, and security is affected by a number of secondary requirements that are described in a separate document [RFC2887]. In summary, these are:

- o Ordering Guarantees. A protocol must offer at least one of either source ordered or unordered delivery guarantees. Support for total ordering across multiple senders is not recommended, as it makes it more difficult to scale the protocol, and can more easily be implemented at a higher level.
- o Receiver Scalability. A protocol should be able to support a "large" number of simultaneous receivers per transport group. A typical receiver set could be on the order of at least 1,000 - 10,000 simultaneous receivers per group, or could even eventually scale up to millions of receivers in the large Internet.
- o Real-Time Feedback. Some versions of RMCC may require soft real-time feedback, so a protocol may provide some means for this information to be measured and returned to the sender. While this does not require that a protocol deliver data in soft real-time, it is an important application requirement that can be provided easily given real-time feedback.
- o Delivery Guarantees. In many applications, a logically defined unit or units of data is to be delivered to multiple clients, e.g., a file or a set of files, a software package, a stock quote or package of stock quotes, an event notification, a set of slides, a frame or block from a video. An application data unit is defined to be a logically separable unit of data that is useful to the application. In some cases, an application data unit may be short enough to fit into a single packet (e.g., an event

Whetten, et al.

Informational

[Page 3]

□

RFC 3048

RMT Building Blocks

January 2001

notification or a stock quote), whereas in other cases an application data unit may be much longer than a packet (e.g., a software package). A protocol must provide good throughput of application data units to receivers. This means that most data that is delivered to receivers is useful in recovering the application data unit that they are trying to receive. A protocol may optionally provide delivery confirmation, i.e., a mechanism for receivers to inform the sender when data has been delivered.

There are two types of confirmation, at the application data unit level and at the packet level. Application data unit confirmation is useful at the application level, e.g., to inform the application about receiver progress and to decide when to stop sending packets about a particular application data unit. Packet confirmation is useful at the transport level, e.g., to inform the transport level when it can release buffer space being used for storing packets for which delivery has been confirmed. Packet level confirmation may also aid in application data unit confirmation.

- o Network Topologies. A protocol must not break the network when deployed in the full Internet. However, we recognize that intranets will be where the first wave of deployments happen, and so are also very important to support. Thus, support for satellite networks (including those with terrestrial return paths or no return paths at all) is encouraged, but not required.
- o Group Membership. The group membership algorithms must be scalable. Membership can be anonymous (where the sender does not know the list of receivers), or fully distributed (where the sender receives a count of the number of receivers, and optionally a list of failures).
- o Example Applications. Some of the applications that a RM protocol could be designed to support include multimedia broadcasts, real time financial market data distribution, multicast file transfer, and server replication.

In the rest of this document the following terms will be used with a specific connotation: "protocol family", "protocol component", "building block", "protocol core", and "protocol instantiation". A "protocol family" is a broad class of RM protocols which share a common characteristic. In our classification, this characteristic is the mechanism used to achieve reliability. A "protocol component" is a logical part of the protocol that addresses a particular functionality. A "building block" is a constituent of a protocol that implements one, more than one or a part of a component. A "protocol core" is the set of functionality required for the

Whetten, et al.

Informational

[Page 4]

□

RFC 3048

RMT Building Blocks

January 2001

instantiation of a complete protocol, that is not specified by any building block. Finally a "protocol instantiation" is an actual RM protocol defined in term of building blocks and a protocol core.

1.1. Protocol Families

The design-space document [RFC2887] also provides a taxonomy of the most popular approaches that have been proposed over the last ten years. After congestion control, the primary challenge has been that of meeting the requirement for ensuring good throughput in a way that scales to a large number of receivers. For protocols that include a back-channel for recovery of lost packets, the ability to take

advantage of support of elements in the network has been found to be very beneficial for supporting good throughput for a large numbers of receivers. Other protocols have found it very beneficial to transmit coded data to achieve good throughput for large numbers of receivers.

This taxonomy breaks proposed protocols into four families. Some protocols in the family provide packet level delivery confirmation that may be useful to the transport level. All protocols in all families can be supplemented with higher level protocols that provide delivery confirmation of application data units.

- 1 NACK only. Protocols such as SRM [FJM95] and MDP2 [MA99] attempt to limit traffic by only using NACKs for requesting packet retransmission. They do not require network infrastructure.
- 2 Tree based ACK. Protocols such as RMTP [LP96, PSLM97], RMTP-II [WBPM98] and TRAM [KCW98], use positive acknowledgments (ACKs). ACK based protocols reduce the need for supplementary protocols that provide delivery confirmation, as the ACKs can be used for this purpose. In order to avoid ACK implosion in scaled up deployments, the protocol can use servers placed in the network.
- 3 Asynchronous Layered Coding (ALC). These protocols (examples include [RV97] and [BLMR98]) use sender-based Forward Error Correction (FEC) methods with no feedback from receivers or the network to ensure good throughput. These protocols also used sender-based layered multicast and receiver-driven protocols to join and leave these layers with no feedback to the sender to achieve scalable congestion control.
- 4 Router assist. Like SRM, protocols such as PGM [FLST98] and [LG97] also use negative acknowledgments for packet recovery. These protocols take advantage of new router software to do constrained negative acknowledgments and retransmissions. Router assist protocols can also provide other functionality more efficiently than end to end protocols. For example, [LVS99] shows

Whetten, et al.

Informational

[Page 5]

□

RFC 3048

RMT Building Blocks

January 2001

how router assist can provide fine grained congestion control for ALC protocols. Router assist protocols can be designed to complement all protocol families described above.

Note that the distinction in protocol families is not necessarily precise and mutually exclusive. Actual protocols may use a combination of the mechanisms belonging to different classes. For example, hybrid NACK/ACK based protocols (such as [WBPM98]) are possible. Other examples are protocols belonging to class 1 through 3 that take advantage of router support.

2. Building Blocks Rationale

As specified in RFC 2357 [MRBP98], no single reliable multicast protocol will likely meet the needs of all applications. Therefore, the IETF expects to standardize a number of protocols that are

tailored to application and network specific needs. This document concentrates on the requirements for "one-to-many bulk-data transfer", but in the future, additional protocols and building-blocks are expected that will address the needs of other types of applications, including "many-to-many" applications. Note that bulk-data transfer does not refer to the timeliness of the data, rather it states that there is a large amount of data to be transferred in a session. The scope and approach taken for the development of protocols for these additional scenarios will depend upon large part on the success of the "building-block" approach put forward in this document.

2.1. Building Blocks Advantages

Building a large piece of software out of smaller modular components is a well understood technique of software engineering. Some of the advantages that can come from this include:

- o Specification Reuse. Modules can be used in multiple protocols, which reduces the amount of development time required.
- o Reduced Complexity. To the extent that each module can be easily defined with a simple API, breaking a large protocol in to smaller pieces typically reduces the total complexity of the system.
- o Reduced Verification and Debugging Time. Reduced complexity results in reduced time to debug the modules. It is also usually faster to verify a set of smaller modules than a single larger module.

Whetten, et al.

Informational

[Page 6]

RFC 3048

RMT Building Blocks

January 2001

- o Easier Future Upgrades. There is still ongoing research in reliable multicast, and we expect the state of the art to continue to evolve. Building protocols with smaller modules allows them to be more easily upgraded to reflect future research.
- o Common Diagnostics. To the extent that multiple protocols share common packet headers, packet analyzers and other diagnostic tools can be built which work with multiple protocols.
- o Reduces Effort for New Protocols. As new application requirements drive the need for new standards, some existing modules may be reused in these protocols.
- o Parallelism of Development. If the APIs are defined clearly, the development of each module can proceed in parallel.

2.2. Building Block Risks

Like most software specification, this technique of breaking down a protocol in to smaller components also brings tradeoffs. After a

certain point, the disadvantages outweigh the advantages, and it is not worthwhile to further subdivide a problem. These risks include:

- o Delaying Development. Defining the API for how each two modules inter-operate takes time and effort. As the number of modules increases, the number of APIs can increase at more than a linear rate. The more tightly coupled and complex a component is, the more difficult it is to define a simple API, and the less opportunity there is for reuse. In particular, the problem of how to build and standardize fine grained building blocks for a transport protocol is a difficult one, and in some cases requires fundamental research.
- o Increased Complexity. If there are too many modules, the total complexity of the system actually increases, due to the preponderance of interfaces between modules.
- o Reduced Performance. Each extra API adds some level of processing overhead. If an API is inserted in to the "common case" of packet processing, this risks degrading total protocol performance.
- o Abandoning Prior Work. The development of robust transport protocols is a long and time intensive process, which is heavily dependent on feedback from real deployments. A great deal of work has been done over the past five years on components of protocols such as RMTP-II, SRM, and PGM. Attempting to dramatically re-engineer these components risks losing the benefit of this prior work.

Whetten, et al.

Informational

[Page 7]

□

RFC 3048

RMT Building Blocks

January 2001

2.3. Building Block Requirements

Given these tradeoffs, we propose that a building block must meet the following requirements:

- o Wide Applicability. In order to have confidence that the component can be reused, it should apply across multiple protocol families and allow for the component's evolution.
- o Simplicity. In order to have confidence that the specification of the component APIs will not dramatically slow down the standards process, APIs must be simple and straight forward to define. No new fundamental research should be done in defining these APIs.
- o Performance. To the extent possible, the building blocks should attempt to avoid breaking up the "fast track", or common case packet processing.

3. Protocol Components

This section proposes a functional decomposition of RM bulk-data protocols from the perspective of the functional components provided to an application by a transport protocol. It also covers some components that while not necessarily part of the transport protocol,

are directly impacted by the specific requirements of a reliable multicast transport. The next section then specifies recommended building blocks that can implement these components.

Although this list tries to cover all the most common transport-related needs of one-to-many bulk-data transfer applications, new application requirements may arise during the process of standardization, hence this list must not be interpreted as a statement of what the transport layer should provide and what it should not. Nevertheless, it must be pointed out that some functional components have been deliberately omitted since they have been deemed irrelevant to the type of application considered (i.e., one-to-many bulk-data transfer). Among these are advanced message ordering (i.e., those which cannot be implemented through a simple sequence number) and atomic delivery.

It is also worth mentioning that some of the functional components listed below may be required by other functional components and not directly by the application (e.g., membership knowledge is usually required to implement ACK-based reliability).

The following list covers various transport functional components and splits them in sub-components.

Whetten, et al.

Informational

[Page 8]

□

RFC 3048

RMT Building Blocks

January 2001

- o Data Reliability (ensuring good throughput)
 - Loss Detection/Notification
 - Loss Recovery
 - Loss Protection
- o Congestion Control
 - Congestion Feedback
 - Rate Regulation
 - Receiver Controls
- o Security
- o Group membership
 - Membership Notification
 - Membership Management
- o Session Management
 - Group Membership Tracking
 - Session Advertisement
 - Session Start/Stop
 - Session Configuration/Monitoring
- o Tree Configuration

Note that not all components are required by all protocols, depending upon the fully defined service that is being provided by the protocol. In particular, some minimal service models do not require many of these functions, including loss notification, loss recovery,

and group membership.

3.1. Sub-Components Definition

Loss Detection/Notification. This includes how missing packets are detected during transmission and how knowledge of these events are propagated to one or more agents which are designated to recover from the transmission error. This task raises major scalability issues and can lead to feedback implosion and poor throughput if not properly handled. Mechanisms based on TRACKs (tree-based positive acknowledgements) or NACKs (negative acknowledgements) are the most widely used to perform this function. Mechanisms based on a combination of TRACKs and NACKs are also possible.

Loss Recovery. This function responds to loss notification events through the transmission of additional packets, either in the form of copies of those packets lost or in the form of FEC packets. The manner in which this function is implemented can significantly affect the scalability of a protocol.

Whetten, et al.

Informational

[Page 9]

□

RFC 3048

RMT Building Blocks

January 2001

Loss Protection. This function attempts to mask packet-losses so that they don't become Loss Notification events. This function can be realized through the pro-active transmission of FEC packets. Each FEC packet is created from an entire application data unit [LMSSS97] or a portion of an application data unit [RV97], [BKKKLZ95], a fact that allows a receiver to recover from some packet loss without further retransmissions. The number of losses that can be recovered from without requiring retransmission depends on the amount of FEC packets sent in the first place. Loss protection can also be pushed to the extreme when good throughput is achieved without any Loss Detection/Notification and Loss Recovery functionality, as in the ALC family of protocols defined above.

Congestion Feedback. For sender driven congestion control protocols, the receiver must provide some type of feedback on congestion to the sender. This typically involves loss rate and round trip time measurements.

Rate Regulation. Given the congestion feedback, the sender then must adjust its rate in a way that is fair to the network. One proposal that defines this notion of fairness and other congestion control requirements is [Whetten99].

Receiver Controls. In order to avoid allowing a receiver that has an extremely slow connection to the sender to stop all progress within single rate schemes, a congestion control algorithm will often require receivers to leave groups. For multiple rate approaches, receivers of all connection speeds can have data delivered to them according to the rate of their connection without slowing down other receivers.

Security. Security for reliable multicast contains a number of

complex and tricky issues that stem in large part from the IP multicast service model. In this service model, hosts do not send traffic to another host, but instead elect to receive traffic from a multicast group. This means that any host may join a group and receive its traffic. Conversely, hosts may also leave a group at any time. Therefore, the protocol must address how it impacts the following security issues:

- o Sender Authentication (since any host can send to a group),
- o Data Encryption (since any host can join a group)
- o Transport Protection (denial of service attacks, through corruption of transport state, or requests for unauthorized resources)

Whetten, et al.

Informational

[Page 10]

□

RFC 3048

RMT Building Blocks

January 2001

- o Group Key Management (since hosts may join and leave a group at any time) [WHA98]

In particular, a transport protocol needs to pay particular attention to how it protects itself from denial of service attacks, through mechanisms such as lightweight authentication of control packets [HW99].

With Source Specific Multicast service model (SSM), a host joins specifically to a sender and group pair. Thus, SSM offers more security against hosts receiving traffic from a denial of service attack where an arbitrary sender sends packets that hosts did not specifically request to receive. Nevertheless, it is recommended that additional protections against such attacks should be provided when using SSM, because the protection offered by SSM against such attacks may not be enough.

Sender Authentication, Data Encryption, and Group Key Management. While these functions are not typically part of the transport layer per se, a protocol needs to understand what ramifications it has on data security, and may need to have special interfaces to the security layer in order to accommodate these ramifications.

Transport Protection. The primary security task for a transport layer is that of protecting the transport layer itself from attack. The most important function for this is typically lightweight authentication of control packets in order to prevent corruption of state and other denial of service attacks.

Membership Notification. This is the function through which the data source--or upper level agent in a possible hierarchical organization--learns about the identity and/or number of receivers or lower level agents. To be scalable, this typically will not provide total knowledge of the identity of each receiver.

Membership Management. This implements the mechanisms for members to

join and leave the group, to accept/refuse new members, or to terminate the membership of existing members.

Group Membership Tracking. As an optional feature, a protocol may interface with a component which tracks the identity of each receiver in a large group. If so, this feature will typically be implemented out of band, and may be implemented by an upper level protocol. This may be useful for services that require tracking of usage of the system, billing, and usage reports.

Whetten, et al.

Informational

[Page 11]

□

RFC 3048

RMT Building Blocks

January 2001

Session Advertisement. This publishes the session name/contents and the parameters needed for its reception. This function is usually performed by an upper layer protocol (e.g., [HPW99] and [HJ98]).

Session Start/Stop. These functions determine the start/stop time of sender and/or receivers. In many cases this is implicit or performed by an upper level application or protocol. In some protocols, however, this is a task best performed by the transport layer due to scalability requirements.

Session Configuration/Monitoring. Due to the potentially far reaching scope of a multicast session, it is particularly important for a protocol to include tools for configuring and monitoring the protocol's operation.

Tree Configuration. For protocols which include hierarchical elements (such as PGM and RMTP-II), it is important to configure these elements in a way that has approximate congruence with the multicast routing topology. While tree configuration could be included as part of the session configuration tools, it is clearly better if this configuration can be made automatic.

4. Building Block Recommendations

The families of protocols introduced in section 1.1 generally use different mechanisms to implement the protocol functional components described in section 3. This section tries to group these mechanisms in macro components that define protocol building blocks.

A building block is defined as

"a logical protocol component that results in explicit APIs for use by other building blocks or by the protocol client."

Building blocks are generally specified in terms of the set of algorithms and packet formats that implement protocol functional components. A building block may also have API's through which it communicates to applications and/or other building blocks. Most building blocks should also have a management API, through which it communicates to SNMP and/or other management protocols.

In the following section we will list a number of building blocks which, at this stage, seem to cover most of the functional components needed to implement the protocol families presented in section 1.1. Nevertheless this list represents the "best current guess", and as such it is not meant to be exhaustive. The actual building block decomposition, i.e., the division of functional components into building blocks, may also have to be revised in the future.

Whetten, et al.

Informational

[Page 12]

RFC 3048

RMT Building Blocks

January 2001

4.1. NACK-based Reliability

This building block defines NACK-based loss detection/notification and recovery. The major issues it addresses are implosion prevention (suppression) and NACK semantics (i.e., how packets to be retransmitted should be specified, both in the case of selective and FEC loss repair). Suppression mechanisms to be considered are:

- o Multicast NACKs
- o Unicast NACKs and Multicast confirmation

These suppression mechanisms primarily need to both minimize delay while also minimizing redundant messages. They may also need to have special weighting to work with Congestion Feedback.

4.2. FEC coding

This building block is concerned with packet level FEC information when FEC codes are used either proactively or as repairs in reaction to lost packets. It specifies the FEC codec selection and the FEC packet naming (indexing) for both reactive FEC repair and pro-active FEC.

4.3. Congestion Control

There will likely be multiple versions of this building block, corresponding to different design policies in addressing congestion control. Two main approaches are considered for the time being: a source-based rate regulation with a single rate provided to all the receivers in the session, and a multiple rate receiver-driven approach with different receivers receiving at different rates in the same session. The multiple rate approach may use multiple layers of multicast traffic [VRC98] or router filtering of a single layer [LVS99]. The multiple rate approach is most applicable for ALC protocols.

Both approaches are still in the phase of study, however the first seems to be mature enough [HFW99] to allow the standardization process to begin.

At the time of writing this document, a third class of congestion control algorithm based on router support is beginning to emerge in the IRTF RMRG [LVS99]. This work may lead to the future standardization of one or more additional building blocks for

congestion control.

Whetten, et al.

Informational

[Page 13]

□

RFC 3048

RMT Building Blocks

January 2001

4.4. Generic Router Support

The task of designing RM protocols can be made much easier by the presence of some specific support in routers. In some application-specific cases, the increased benefits afforded by the addition of special router support can justify the resulting additional complexity and expense [FLST98].

Functional components which can take advantage of router support include feedback aggregation/suppression (both for loss notification and congestion control) and constrained retransmission of repair packets. Another component that can take advantage of router support is intentional packet filtering to provide different rates of delivery of packets to different receivers from the same multicast packet stream. This could be most advantageous when combined with ALC protocols [LVS99].

The process of designing and deploying these mechanisms inside routers can be much slower than the one required for end-host protocol mechanisms. Therefore, it would be highly advantageous to define these mechanisms in a generic way that multiple protocols can use if it is available, but do not necessarily need to depend on.

This component has two halves, a signaling protocol and actual router algorithms. The signaling protocol allows the transport protocol to request from the router the functions that it wishes to perform, and the router algorithms actually perform these functions. It is more urgent to define the signaling protocol, since it will likely impact the common case protocol headers.

An important component of the signaling protocol is some level of commonality between the packet headers of multiple protocols, which allows the router to recognize and interpret the headers.

4.5. Tree Configuration

It has been shown that the scalability of RM protocols can be greatly enhanced by the insertion of some kind of retransmission or feedback aggregation agents between the source and receivers. These agents are then used to form a tree with the source at (or near) the root, the receivers at the leaves of the tree, and the aggregation/local repair nodes in the middle. The internal nodes can either be dedicated software for this task, or they may be receivers that are performing dual duty.

The effectiveness of these agents to assist in the delivery of data is highly dependent upon how well the logical tree they use to communicate matches the underlying routing topology. The purpose of

Whetten, et al.

Informational

[Page 14]

□

RFC 3048

RMT Building Blocks

January 2001

this building block would be to construct and manage the logical tree connecting the agents. Ideally, this building block would perform these functions in a manner that adapts to changes in session membership, routing topology, and network availability.

4.6. Data Security

At the time of writing, the security issues are the subject of research within the IRTF Secure Multicast Group (SMuG). Solutions for these requirements will be standardized within the IETF when ready.

4.7. Common Headers

As pointed out in the generic router support section, it is important to have some level of commonality across packet headers. It may also be useful to have common data header formats for other reasons. This building block would consist of recommendations on fields in their packet headers that protocols should make common across themselves.

4.8. Protocol Cores

The above building blocks consist of the functional components listed in section 3 that appear to meet the requirements for being implemented as building blocks presented in section 2.

The other functions from section 3, which are not covered above, should be implemented as part of "protocol cores", specific to each protocol standardized.

5. Security Considerations

RFC 2357 specifically states that "reliable multicast Internet-Drafts reviewed by the Transport Area Directors must explicitly explore the security aspects of the proposed design." Specifically, RMT building block works in progress must examine the denial-of-service attacks that can be made upon building blocks and affected by building blocks upon the Internet at large. This requirement is in addition to any discussions regarding data-security, that is the manipulation of or exposure of session information to unauthorized receivers. Readers are referred to section 5.e of RFC 2357 for further details.

6. IANA Considerations

There will be more than one building block, and possibly multiple versions of individual building blocks as their designs are refined. For this reason, the creation of new building blocks and new building block versions will be administered via a building block registry

Whetten, et al.

Informational

[Page 15]

that will be administered by IANA. Initially, this registry will be empty, since the building blocks described in sections 4.1 to 4.3 are presented for example and design purposes. The requested IANA building block registry will be populated from specifications as they are approved for RFC publication (using the "Specification Required" policy as described in RFC 2434 [RFC2434]). A registration will consist of a building block name, a version number, a brief text description, a specification RFC number, and a responsible person, to which IANA will assign the type number.

7. Conclusions

In this document, we briefly described a number of building blocks that may be used for the generation of reliable multicast protocols to be used in the application space of one-to-many reliable bulk-data transfer. The list of building blocks presented was derived from considering the functions that a protocol in this space must perform and how these functions should be grouped. This list is not intended to be all-inclusive but instead to act as guide as to which building blocks are considered during the standardization process within the Reliable Multicast Transport WG.

8. Acknowledgements

This document represents an overview of a number of building blocks for one to many bulk data transfer that may be ready for standardization within the RMT working group. The ideas presented are not those of the authors, rather they are a summarization of many years of research into multicast transport combined with the varied presentations and discussions in the IRTF Reliable Multicast Research Group. Although they are too numerous to list here, we thank everyone who has participated in these discussions for their contributions.

9. References

- [BKKKLZ95] J. Bloemer, M. Kalfane, M. Karpinski, R. Karp, M. Luby, D. Zuckerman, "An XOR-based Erasure Resilient Coding Scheme," ICSI Technical Report No. TR-95-048, August 1995.
- [BLMR98] J. Byers, M. Luby, M. Mitzenmacher, A. Rege, "A Digital Fountain Approach to Reliable Distribution of Bulk Data," Proc ACM SIGCOMM 98.
- [FJM95] S. Floyd, V. Jacobson, S. McCanne, "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing," Proc ACM SIGCOMM 95, Aug 1995 pp. 342-356.

- [FLST98] D. Farinacci, S. Lin, T. Speakman, and A. Tweedly, "PGM reliable transport protocol specification," Work in Progress.
- [HFW99] M. Handley, S. Floyd, B. Whetten, "Strawman Specification for TCP Friendly (Reliable) Multicast Congestion Control (TFMCC)," Work in Progress.
- [HJ98] Handley, M. and V. Jacobson, "SDP: Session Description Protocol", RFC 2327, April 1998.
- [HPW99] M. Handley, C. Perkins, E. Whelan, "Session Announcement Protocol," Work in Progress, June 1999.
- [HW99] T. Hardjorno, B. Whetten, "Security Requirements for RMTP-II," Work in Progress, June 1999.
- [RFC2887] Handley, M., Whetten, B., Kermode, R., Floyd, S., Vicisano, L. and M. Luby, "The Reliable Multicast Design Space for Bulk Data Transfer", RFC 2887, August 2000.
- [KCW98] M. Kadansky, D. Chiu, and J. Wesley, "Tree-based reliable multicast (TRAM)," Work in Progress.
- [Kermode98] R. Kermode, "Scoped Hybrid Automatic Repeat Request with Forward Error Correction," Proc ACM SIGCOMM 98, Sept 1998.
- [LDW98] M. Lucas, B. Dempsey, A. Weaver, "MESH: Distributed Error Recovery for Multimedia Streams in Wide-Area Multicast Networks".
- [LESZ97] C-G. Liu, D. Estrin, S. Shenkar, L. Zhang, "Local Error Recovery in SRM: Comparison of Two Approaches," USC Technical Report 97-648, Jan 1997.
- [LG97] B.N. Levine, J.J. Garcia-Luna-Aceves, "Improving Internet Multicast Routing with Routing Labels," IEEE International Conference on Network Protocols (ICNP-97), Oct 28-31, 1997, p.241-250.
- [LP96] K. Lin and S. Paul. "RMTP: A Reliable Multicast Transport Protocol," IEEE INFOCOMM 1996, March 1996, pp. 1414-1424.
- [LMSSS97] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, V. Stemann, "Practical Loss-Resilient Codes", Proc ACM Symposium on Theory of Computing, 1997.

Whetten, et al.

Informational

[Page 17]

□

RFC 3048

RMT Building Blocks

January 2001

- [LVS99] M. Luby, L. Vicisano, T. Speakman. "Heterogeneous multicast congestion control based on router packet filtering", RMT working group, June 1999, Pisa, Italy.

- [MA99] J. Macker, B. Adamson. "Multicast Dissemination Protocol version 2 (MDPv2)," Work in Progress, <http://manimac.itd.nrl.navy.mil/MDP>
- [MRBP98] Mankin, A., Romanow, A., Brander, S. and V. Paxson, "IETF Criteria for Evaluating Reliable Multicast Transport and Application Protocols", RFC 2357, June 1998.
- [RFC2434] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 2434, October 1998.
- [OXB99] O. Ozkasap, Z. Xiao, K. Birman. "Scalability of Two Reliable Multicast Protocols", Work in Progress, May 1999.
- [PSLB97] "Reliable Multicast Transport Protocol (RMTP)," S. Paul, K. K. Sabnani, J. C. Lin, and S. Bhattacharyya, IEEE Journal on Selected Areas in Communications, Vol. 15, No. 3, April 1997.
- [RV97] L. Rizzo, L. Vicisano, "A Reliable Multicast Data Distribution Protocol Based on Software FEC Techniques," Proc. of The Fourth IEEE Workshop on the Architecture and Implementation of High Performance Communication Systems (HPCS'97), Sani Beach, Chalkidiki, Greece June 23-25, 1997.
- [VRC98] L. Vicisano, L. Rizzo, J. Crowcroft, "TCP-Like Congestion Control for Layered Multicast Data Transfer", Proc. of IEEE Infocom'98, March 1998.
- [WBPM98] B. Whetten, M. Basavaiah, S. Paul, T. Montgomery, N. Rastogi, J. Conlan, and T. Yeh, "THE RMTP-II PROTOCOL," Work in Progress.
- [WHA98] D. Wallner, E. Hardler, R. Agee, "Key Management for Multicast: Issues and Architectures," Work in Progress.
- [Whetten99] B. Whetten, "A Proposal for Reliable Multicast Congestion Control Requirements," Work in Progress. <http://www.talarian.com/rmtp-ii/overview.htm>

Whetten, et al.

Informational

[Page 18]

RFC 3048

RMT Building Blocks

January 2001

10. Authors' Addresses

Brian Whetten
 Talarian Corporation,
 333 Distel Circle,
 Los Altos, CA 94022, USA

EEmail: whetten@talarian.com

Lorenzo Vicisano
Cisco Systems,
170 West Tasman Dr.
San Jose, CA 95134, USA

EMail: lorenzo@cisco.com

Roger Kermode
Motorola Australian Research Centre
Level 3, 12 Lord St,
Botany NSW 2019, Australia

EMail: Roger.Kermode@motorola.com

Mark Handley, Sally Floyd
ATT Center for Internet Research at ICSI,
International Computer Science Institute,
1947 Center Street, Suite 600,
Berkeley, CA 94704, USA

EMail: mjh@aciri.org, floyd@aciri.org

Michael Luby
600 Alabama Street
San Francisco, CA 94110
Digital Fountain, Inc.

EMail: luby@digitalfountain.com

Whetten, et al.

Informational

[Page 19]

□

RFC 3048

RMT Building Blocks

January 2001

11. Full Copyright Statement

Copyright (C) The Internet Society (2001). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other

Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.

An Application Level Multicast Architecture for Multimedia Communications in the Internet

Kang-Won Lee, Sungwon Ha, Jia-Ru Li, Vaduvur Bharghavan
Coordinated Science Laboratory, University of Illinois at Urbana-Champaign
Email: {kwlee, s-ha, juru, bharghav}@timely.crhc.uiuc.edu

Abstract—In this paper, we propose an application level multicast architecture for multimedia data communications called MHPF (Multicast Heterogeneous Packet Flows), which provides the following services:

- MHPF provides an overlay network that enables application level multicast over the Internet.
- MHPF efficiently supports heterogeneity in the multicast group by ensuring that each receiver receives the highest priority portion of the multimedia data stream that its connection quality can sustain.
- In order to support interactive applications, MHPF provides for loose synchronization among the receivers, where each receiver perceives approximately the same progress of the multicast session.
- At the same time, MHPF guarantees the delivery of the packets in the multimedia data stream that are marked as reliable by the application.

Unlike most contemporary approaches, MHPF achieves these features *without* decomposing the heterogeneous multimedia stream into component layers; instead it handles the multimedia stream as a single heterogeneous data stream with the aid of HPF transport protocol at the end hosts and the special functionality at the designated MHPF multicast servers. We present a set of simulation-based performance results to illustrate that MHPF achieves its design goals.

I. INTRODUCTION

In recent years, interactive and real-time multimedia applications such as video teleconferencing, multimedia streaming, distributed shared whiteboard, and have become increasingly popular in the Internet. The emergence of such applications has created two new phenomena in the Internet: (a) *multicast* support is becoming critical, since such applications typically have multiple participants, and (b) the data streams are becoming *heterogeneous* in nature (e.g. a typical video conference session consists of text data, multi-resolution audio and video streams with different reliability and priority requirements). Our target set of applications thus requires sophisticated multimedia support for small to medium multicast groups.

Supporting multicast communications of interactive multimedia applications thus introduces unique challenges hitherto unaddressed in traditional network architectures and transport protocols:

- Multicast groups consist of *heterogeneous* receivers, i.e. different receivers in the same multicast group may have

different processing capability and may perceive different connection qualities.

- Multimedia data streams are often *multi-resolution*, consisting of interleaved sub-streams with different priority requirements, where lower priority streams progressively improve the perceived data quality at the receiver, e.g. I/P/B frames of an MPEG stream.

Taking the environment and application requirements into consideration, we have identified three *goals* that our architecture for multicast heterogeneous streams must satisfy:

1. Different receivers of a multicast group may perceive different connection qualities, and *each receiver must receive the highest priority portion of the heterogeneous data stream that its connection quality can sustain.*
2. The receivers in the same group must be loosely synchronized, i.e. they must perceive approximately the same progress of the multimedia session. In particular, *it is preferable for a poorly connected receiver to lose more frames but be synchronized with other receivers rather than receive all the frames and lag behind.*
3. In addition, all the packets that are marked reliable in the heterogeneous packet stream by the application need to be delivered reliably to all the receivers.

These three characteristics serve as our metrics for evaluating existing solutions, and as guidelines for the design of our multicast architecture.

The standard IP multicast protocol only provides best-effort delivery of multicast data. Recent transport layer proposals designed to address at least some of the issues listed above include RTP [1], DSG, [2], RLM [3] (for real-time/streaming data delivery), and a variety of (semi-) reliable multicast transport protocols [4], [5] (for error-resilient delivery).

In related work, a *layered multicast* approach is becoming very popular for multicasting prioritized data streams in the Internet [3], [6], [7], [8], [9], [10]. Essentially, the approach is to decompose the *multi-resolution* data stream (e.g. an MPEG stream) into component *single resolution* streams (e.g. I frame, P frame, and B frame streams), establish a distinct multicast group for each component stream using standard IP multicast, and let each receiver independently decide which multicast groups it wants to join based on the perceived connection quality. This approach has the charm of being intuitively simple, and of

not requiring special mechanisms other than standard IP multicast functionality from the network. However, it also has several inherent limitations imposed by layering such as the coarse granularity of adaptation to network dynamics, potentially slow reaction to changes in connection quality, and destructive interference by concurrent adaptation by multiple receivers [8], [9], [10]. We revisit these issues in the next section.

In this paper, we present the MHPF (Multicast Heterogeneous Packet Flows) architecture that supports multicast communications of heterogeneous data streams for small to medium multicast groups without decomposing them into component homogeneous data streams. MHPF exhibits the three elements of the 'desired' behavior of the multimedia multicast communications mentioned above without incurring the drawbacks of the layered approach, but at the expense of introducing additional functionality in specialized "MHPF servers." MHPF is composed of two key components: (a) an adaptive transport protocol called HPF (Heterogeneous Packet Flows) [12], which provides end-to-end transport of heterogeneous packet flows, and (b) a virtual network of collaborating MHPF servers, which provides application level multicast support mechanisms for heterogeneous packet flows. The focus of this paper is the latter component.

In summary, related work has typically taken a minimalist approach, and tried to answer the question: how can we best support multimedia multicast communications without adding any special mechanisms in the network? We approach the problem from the opposite perspective: what are the minimum mechanisms that need to be implemented by specialized multicast servers in the overlay network in order to support the desired service for multicast multimedia communications? To this end, our first goal is to achieve the desired service for multimedia multicast identified above; subject to this goal, our second goal is to minimize the state management and computation complexity of the mechanisms in the multicast servers. Finally, we compare the two approaches to see whether we are able to achieve justifiable performance improvements at acceptable cost. In order to achieve incremental deployability over the current Internet, our approach is to enable specialized multicast services in "application level" MHPF-aware servers that provide an overlay network. At the outset, we recognize that the mechanisms proposed in this paper can be instantiated either in the multicast routers themselves or in the application level servers. Our choice has been motivated by engineering and deployability concerns.

The remainder of the paper is organized as follows: Section II presents the related work. Section III presents the service model and the design choices for the MHPF architecture. Section IV details the design of the MHPF architecture focusing on the network level functionality. Section V presents a simulation-based performance evaluation.

Section VI summarizes the issues and trade-offs.

II. RELATED WORK

Since multi-resolution streaming is becoming very popular in the multimedia coding community, future networks must support heterogeneous multicast data streams to enable multi-party multimedia applications. Related work in this area can be categorized in the following three ways: (a) sender-based single rate multicast approach [14], (b) replicated multiple multicast group approach [2], and (c) layered multicast approach [3], [6], [7], [8], [9], [10], [11].

The first approach is limited since the rate control by the sender often reacts slowly to network dynamics and cannot provide optimal performance in the multicast environment due to the *inter-receiver fairness problem* [15], and hence violates the first point of our goals. The second approach uses multiple *replicated* video streams with different quality multicast to distinct groups and lets each receiver decide which group to join based on its connection capability. Although this scheme is free from the inter-receiver fairness issue, the bandwidth efficiency may be poor when there are multiple *replicated* streams on the same path, i.e. inefficient achieving the first goal of the desired system. Hence most contemporary work adopts the third approach.

In layered multicasting [3], the sender decomposes the video stream into multiple component video substreams (or layers), so that the lowest layer provides the low resolution (and high priority) data and the additional layer progressively improves the picture quality, and establishes a distinct multicast group for each component stream. Each receiver independently decides how many multicast groups to join according to its connection quality: when a receiver sees a congestion it drops the highest layer and when it does not see congestions for a while it joins the next layer. This approach has the charm of being intuitively simple and efficient in terms of network bandwidth usage since layers are cumulative, and of not requiring any special support mechanism other than standard IP multicast routing functionality from the network.

However, researchers have found a list of problems with layered multicasting [8], [9], [10], [11]: (a) coarse grain rate adaptation due to join/leave latency,¹ (b) bandwidth fluctuation due to relatively high bandwidth layers, (c) destructive interference of independent join/leave operations of each receiver, (d) priority-based delivery is not guaranteed (especially during the join experiments of its own or other receivers), (e) process overhead of decomposition and resynchronization of the video stream at the end hosts,

¹Join operations are relatively fast if the local multicast router has already joined the group; otherwise, join operations are expensive taking up to one round-trip time to the sender's domain or a special domain called *root domain* [13]. Leave latencies are also relatively high. In IGMP version 1, leave operations take a couple of minutes. In version 2, it is configurable using the *max response time* field in the order of 100 msec.

(f) unnecessary coupling of video encoding and rate adaptation mechanisms (e.g. the number of layers and their bandwidths are determined by video encoding scheme), (g) overhead of *shared learning* [3] in terms of message exchange and state management, and (h) congestion information acquired from *shared learning* is often incorrect and misleads the decision of each receiver. Points (a), (b), (c), and (d) are related to the problem of layering, and points (e), (f), (g) and (h) are from pushing most functionality to the application.

Some researchers [8], [9], [10] tried to alleviate the problem incurred by independent join/leave operations via synchronizing the receiver actions (point c) [8], [9], or structuring and sharing the rate adaptation history of individual receiver (point g, h) [10]. Wu et al. [9] proposed using small equal bandwidth layers called “ThinStreams” in order to alleviate the bandwidth fluctuation problem (point b) and to decouple control and video encoding mechanism (point f). However, we claim that incremental enhancements cannot solve all the problems of layered multicast approach since some of the problems are inherent to the ‘layering mechanism’ itself. In summary, layering satisfies the second goal of the desired system, but is inefficient achieving the first goal and does not address the third point.

In related work, we proposed an adaptive transport protocol called HPF, which provides end-to-end unicast transport for heterogeneous packet flows [12]. The basic idea is that rather than using discrete layers, the sender *interleaves packets with different priority and reliability requirements in a single transport layer stream* and enables the receiver to receive as much high priority packets as the connection quality can sustain. HPF allows an application to specify per-frame policies for reliability, priority, and deadline requirements, and implements the end-to-end framing, flow control, error control, and congestion control mechanisms for the heterogeneous data stream.

The MHPF architecture proposed in this paper extends the idea of HPF to the multicast domain. Specifically, MHPF adopts the HPF transport protocol as an end-to-end mechanism for multicasting heterogeneous data streams and introduces special mechanisms, such as priority-based filtering, in the MHPF servers to support efficient transmission of heterogeneous data streams in the network. As a result, MHPF can adapt its sending rate at a much finer grain to the network dynamics than the layering mechanisms can.

By the design choice of the MHPF architecture, we do not require ‘layering’ in order to support efficient multicasting of prioritized multimedia data streams; hence MHPF does not suffer from the problems of layered multicast approach listed above. However, this comes at the price of establishing an overlay network for enabling application level multicast with specialized functionality to support multimedia streams. We explore the trade-offs of the

improved service model of MHPF and its computational complexity later in the paper.

III. MHPF OVERVIEW

Let us now look at the service model of MHPF, and the mechanisms needed to achieve the service model. To recapitulate, the target applications of MHPF are multi-party multimedia applications that transmit multi-resolution data streams with different reliability and priority requirements.

A. Service Model

The service model for heterogeneous data stream multicast needs to address two aspects: reliability semantics, and synchronization among receivers.

In terms of reliability, strict TCP-like semantics of guaranteeing sequencing and reliability for all packets is neither necessary nor efficient for typical multicast multimedia traffic. However, the heterogeneous nature of the multimedia stream requires some portion of the data to be delivered reliably at the receivers, e.g. control packets in an MPEG stream.

Providing semi-reliable semantics can be done in two ways: deliver every packet with a certain probability, or guarantee reliable delivery for certain designated packets and best-effort delivery for the other packets with preferential delivery of higher priority packets. We choose the latter reliability semantics for MHPF, which we call *partial reliability*.

In order to support interactive multimedia applications, multicast transport protocols need to provide loose synchronization among the receivers in the same multicast group. However, slow receivers cannot keep phase with faster receivers without losing more packets. With the partial reliability semantics, our service model mandates the preferential dropping of low priority best-effort packets to enable MHPF to provide receiver synchronization. Thus, the receiver of an MHPF stream will be provided the abstraction of a single sequenced data stream, possibly with *holes* corresponding to the dropped unreliable low priority packets.

In summary, the service model of MHPF provides partial reliability with preferential delivery of higher priority packets if the connection quality cannot sustain the entire data stream, adapting at a fine grain to the network dynamics, sequencing of all packets, and loose synchronization among receivers.

Now the question is: what are the minimum mechanisms to achieve the MHPF service model? As we mentioned in Section II, HPF provides the end-to-end mechanisms for the partial reliability (i.e. interleaving of reliable and unreliable packets) with flow control, error control, and congestion control for unicast communications. However, the MHPF service model cannot be achieved using only the end-to-end mechanisms of HPF. Specifically,

we want each receiver to receive as much data as its connection can sustain – this means that the sender must be informed of the capacity of the fastest path in the multicast tree and adapt its sending rate to the fastest path capacity. In addition, for those paths that cannot sustain the full data rate, the MHPF service model requires the delivery of the highest priority portion of the heterogeneous stream that can be sustained on the paths – this means that MHPF servers in the network must have the ability to filter data packets based on priority.

In the next section, we identify what are the minimum network level mechanisms required at the MHPF servers to achieve the desired service model.

B. Design Choices for Multicast Multimedia Support

We now consider the design choices using Figure 1 as a reference. In the figure, there is one sender S and four receivers $R_1 - R_4$. The available link bandwidths on each link are shown. The multicast data stream has an aggregate rate of 3 Mbps, with 0.1 Mbps of priority 0 reliable traffic, 0.7 Mbps of priority 1 traffic, 1.0 Mbps of priority 2 traffic, and 1.2 Mbps of priority 3 traffic (where priority level 0 is the highest and 3 is the lowest).

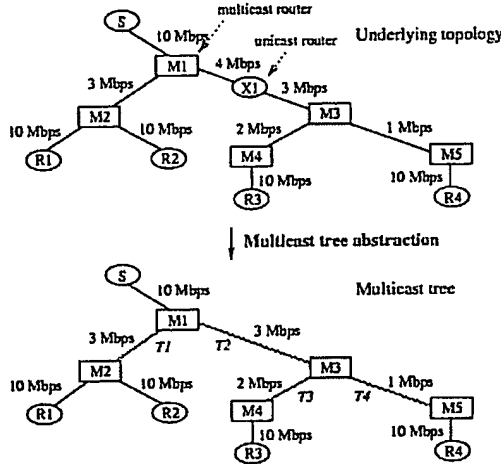


Fig. 1. Example configuration

Ideally, we want the network to provide the following services:

- Each receiver must receive the data stream at the maximum rate that it can sustain at any time. In Figure 1, R_1 and R_2 must receive 3 Mbps, R_3 must receive 2 Mbps, and R_4 must receive 1 Mbps of the heterogeneous data stream.
- Each receiver must preferentially receive higher priority packets over lower priority packets. For example, R_4 must receive 0.1 Mbps of the priority 0 traffic, 0.7 Mbps of priority 1 traffic, and 0.2 Mbps of priority 2 traffic.
- The network should be utilized efficiently. Specifically, a packet should not be forwarded to a multicast router unless there is at least one receiver in its subtree that will receive the packet. For example, the rate on tunnel T_2 should

be 2 Mbps even though the tunnel T_2 can sustain 3 Mbps, because none of the receivers in the subtree of M_3 can sustain more than 2 Mbps.

- As a corollary to the previous goal, the sender should transmit data at the maximum rate that can be sustained among all receivers. In Figure 1, S should transmit at 3 Mbps. Additionally, the transmission rate of the packets marked reliable must be upper-bounded by the minimum rate among all the receivers. In Figure 1, the peak rate for the reliable component of the heterogeneous data stream can be 1 Mbps.

One simple solution to achieve these goals would be to provide end-to-end feedback and priority dropping at the multicast routers. Unfortunately, this approach requires a maintenance of session state in multicast routers. To alleviate this problem, we logically replace the multicast routers in the figure with application level multicast servers.

Further, for the scalability of the architecture, we do not want the sender to maintain per-receiver state. Thus, feedback from the receivers must be aggregated as it trickles back up the (application level) tree, similar to ACK-aggregation in reliable multicast protocols [16]. Specifically, each multicast server can notify its parent about the maximum and minimum sustainable rates among all receivers in its subtree. The sender will then upper-bound the aggregate rate by the maximum rate feedback, and the reliable rate by the minimum rate feedback. Priority dropping at each multicast server will ensure that if the queue on an outgoing link is full and congestion is detected, lower priority packets are preferentially dropped, and higher priority packets get through.

The above solution almost works, but not quite. First, consider tunnel T_2 in Figure 1. M_1 will send 3 Mbps on T_2 in the solution described above. However, M_3 will drop 1 Mbps on T_3 , and 2 Mbps on T_4 . In other words, M_1 should have throttled the transmission rate on T_2 to 2 Mbps in order to ensure that the network is efficiently used. Now consider a more serious problem. The available bandwidth on T_2 is 3 Mbps. However, the outgoing link from M_1 for T_2 has 4 Mbps. Thus, M_1 will pump packets out on T_2 at a rate of 4 Mbps, and packets will be dropped in X_1 . Since we cannot assume any special mechanisms in X_1 , the dropping policy at X_1 is most likely tail-drop, and lower priority packets are not preferentially dropped. Thus, R_3 and R_4 will receive 2 Mbps and 1 Mbps, respectively, but not necessarily the highest priority data.

Both the problems described above can be solved only with some kind of rate adaptation in the overlay tree (similar to the backpressure mechanisms described in [17], [18]). For example, M_1 must estimate the sustainable rate on T_2 , and also get feedback about the maximum rate that M_3 can sustain. M_1 must then throttle the transmission rate on T_2 to the minimum of these values. It is straightforward to see that with a combination of end-to-end feed-

back, priority dropping, and rate adaptation at both the multicast servers and the end host, all the goals of network level mechanism are achieved. This clearly imposes a considerable amount of overhead on the MHPF design compared to layered multicast approach, but must be traded-off against the much improved service model.

IV. DESIGN OF THE MHPF ARCHITECTURE

In this section, we first present an overview of the MHPF architecture using an example. We then present the details of the architecture with the emphasis on the rate adaptation mechanism.

Briefly, MHPF works as follows: For each session, MHPF abstracts a multicast tree T composed only of MHPF servers and multicast tunnels between them, generated by the IP multicast routing. When an application sends a frame, it specifies the reliability and priority parameters for the frame. The HPF protocol at the sender converts the frame into a sequence of one or more packets, all with the same parameters, then queues the packets for transmission in a single heterogeneous data stream. The MHPF servers implement a specialized packet forwarding behavior, so that only the highest priority packets that can be accommodated on a path downstream are transmitted along the path.

Each receiver periodically generates rate feedback at the prescribed time granularity called *epoch*. The rate feedback contains the *number of packets received in the last epoch*, which gives information on the connection quality that each receiver perceives. The feedback from the receivers travels upstream along the same multicast tree T but in the opposite direction, and gets aggregated at the MHPF servers.

The feedback aggregation is done in such a way that when the feedback reaches the sender, it contains the information on the available bandwidth on the fastest and the slowest paths in the multicast tree. The HPF protocol at the sender controls the sending rate of the aggregate heterogeneous stream by the fastest path feedback, and the reliable data rate by the slowest path feedback. The MHPF servers also use the rate feedback in order to provide network level rate adaptation as described in Section III-B.

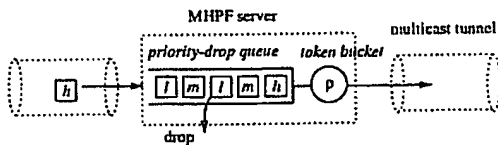


Fig. 2. MHPF server and tunnel abstraction

Figure 2 presents an abstraction of the MHPF server functionality. Basically, it performs rate control on the downstream multicast tunnel using a token bucket, with priority-drop buffer management policy. In the example, a

```

1  Variables at the tunnel source i
2   $\rho_i$  // sending rate of  $i$  in the current epoch
3   $\rho_g$  // sending rate of parent tunnel source  $g$  in the current epoch
4   $\rho_N$  // new downstream (estimated) bottleneck rate for next epoch
5   $sent$  // number of packets sent on  $i$  in the current epoch
6   $recv$  // number of packets received by the receiver connected via
7         // the fastest path in the subtree in the current epoch

8  At the start of each epoch
9  The tunnel source of parent MHPF server  $g$  informs its sending rate
10 The tunnel source  $i$  updates  $\rho_g$ 
11 After this update, the tunnel source  $i$  performs
12   if ( $\rho_g < \rho_i - \alpha$ )
13     increase_constant  $\leftarrow 0$ 
14   else
15     increase_constant  $\leftarrow 1$ 

16 At the end of each epoch
17 The tunnel source  $i$  receives  $min\_recv$ ,  $max\_recv$  and  $\rho_N$  from the child
18  $recv \leftarrow max\_recv$ 
19 if ( $recv = sent$ ) // linear increase
20    $\rho_i \leftarrow \min(\rho_N, \rho_i + increase\_constant)$ 
21 else // multiplicative decrease
22    $\rho_i \leftarrow \min(\rho_N, \rho_i \times 0.5)$ 

```

Fig. 4. Pseudo code of the rate adaptation at the MHPF server

low priority packet in the queue is dropped to accommodate an incoming high priority packet. Figure 3 illustrates an example of the interaction of the MHPF servers and the end hosts, and the protocol structure of the MHPF architecture.

At this point, the high level architecture of MHPF should be fairly clear to the reader. We now present the design details of each component of the MHPF architecture: network level rate adaptation, priority-based packet dropping.

A. Rate Adaptation Mechanism

Rate adaptation occurs in MHPF over discrete periods of time called *epochs* on each downstream multicast tunnel. For simplicity, we assume that the rate adaptation is performed for each multicast session in this section. Later we relax this condition and describe how rate adaptation is performed for aggregate multicast sessions on the tunnel.

When the sender transmits a packet, it inserts the current *epoch id* in the data packet. Each receiver maintains a count of the number of packets it received corresponding to each epoch. When the receiver receives a data packet with a new epoch id, it generates a rate feedback containing max_recv , min_recv , $epoch$, where both max_recv and min_recv contain the number of received packets during the last epoch. The rate feedback is propagated back to the sender along the reverse direction of the multicast tree T .

When an MHPF server receives rate feedback on one of its downstream tunnel, it performs the rate control on the tunnel, according to the rate adaptation algorithm shown in Figure 4 (see also Figure 5). Essentially the rate adaptation algorithm is the popular congestion control paradigm of linear increase multiplicative decrease (LIMD). The basic idea is simple: if there was no packet loss in the last epoch (on the fastest path in the subtree rooted at the tun-

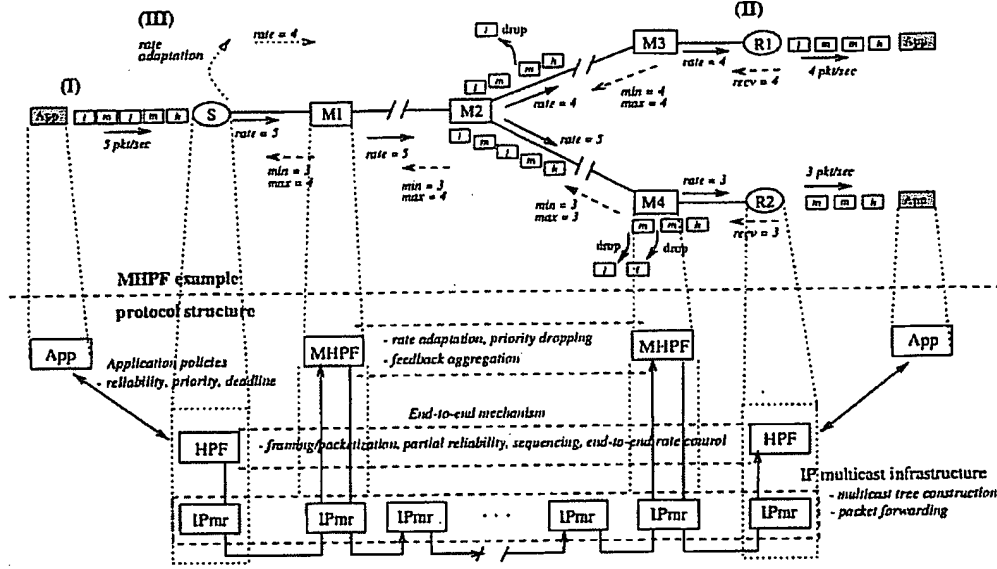


Fig. 3. A simple example of MHPF operation and the protocol structure of the MHPF architecture: Initially, the sender sends at the rate of 5 packets with high/medium/low/medium/low priorities. Let us assume only high priority packets are reliable. Since the $M_2 \rightarrow M_3$ cannot sustain the full data rate, M_2 drops a low priority packet and R_1 receives only 4 packets. Likewise, R_2 only receives 3 packets (stage I, solid lines). The rate feedback from each receiver contains the number received packets and is aggregated at the MHPF servers. When the sender receives the feedback, it contains the numbers of received packets by the fastest ($max = 4$) and the slowest receiver ($min = 3$) in the group (stage II, dashed lines). Using this information, the sender adjusts the aggregate data rate, which becomes 4 in the next round, as well as the reliable data rate, which is unchanged (stage III, dotted lines).

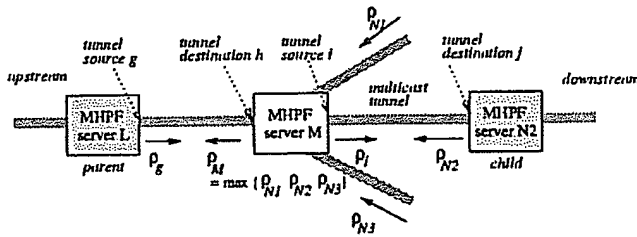


Fig. 5. Information exchange for rate adaptation

nel source i), i.e. $sent = recv$, then increase the sending rate of the tunnel to probe for more available bandwidth in the subtree; otherwise, detect congestion and decrease the sending rate.

The algorithm presented in Figure 4 has additional conditions so that the rate control algorithm enables the tunnel source to quickly synchronize its sending rate with the downstream bottleneck sending rate² by taking the minimum of the calculated sending rate and the estimated downstream bottleneck rate (Figure 4, lines 19 – 22).

In addition, the introduction of ‘increase.flag’ ensures that the sending rate of a tunnel source i does not increase

²The bottleneck sending rate on a path p is the sending rate of a tunnel source k whose immediate downstream multicast tunnel is the bottleneck on the path p . The downstream bottleneck sending rate at a tunnel source i is the bottleneck sending rate on the fastest path in the subtree of the tunnel source i .

without bound when the bottleneck tunnel is located upstream³ (Figure 4, lines 12 – 15, line 20). The constant $\alpha \geq 0$ controls the amount of bandwidth that the tunnel source i can increase over the estimated sending rate of the upstream bottleneck, and is usually set to 0. The reason for controlling the sending rate based on max_recv is because the tunnel source needs to be able to serve the receivers connected via the fastest path in its subtree (Figure 4, line 18).

The state information maintained at an MHPF server M is as follows: parent and children in the multicast tree; the current sending rate of the parent (p_g); for each tunnel source i : the minimum (min_recv) and the maximum (max_recv) number of packets received by the receivers in its subtree in the last epoch, the number of packets sent on the tunnel in the last epoch ($sent$), current sending rate on the tunnel (p_i), and current sending rate of the child (p_N). The sending rate of the parent tunnel source p_g is informed by the parent MHPF server when it updates the sending rate of the tunnel. The MHPF server learns the other information from the feedback from its children.

For the rate feedback aggregation, the MHPF server calculates the minimum of all the min_recv values and the maximum of all the max_recv values from its children and

³In other words, the bottleneck sending rate on the path from the sender to the tunnel source i is smaller than the downstream bottleneck sending rate of i .

forwards this information upstream, along with the maximum of the sending rates on the downstream tunnels, i.e. $\rho_M = \max\{\rho_i\}, \forall i \in \text{downstream tunnel sources}$. After the aggregation, the \min_recv and \max_recv values contain the number of received packets (during the last epoch) by the receivers connected via the slowest and the fastest path in the subtree (rooted at the MHPF server i), respectively. This provides the information for rate adaptation procedure performed by the sender and the upstream MHPF servers.

B. Partial Reliability Mechanism

In order to achieve the partial reliability semantics (with receiver synchronization), the sender needs to control the sending rate of the aggregate data stream (ρ_S) according to the fastest path rate, and the reliable data rate (ρ_R) according to the slowest path rate in the multicast tree. The rationale for the former is to exploit the bandwidth on the fastest path. The rationale for the latter is to avoid reliable packet drops, which will trigger packet retransmission, hence degrades the performance. In addition, we want to preserve the sequencing of interleaved reliable and unreliable packets. In this section, we describe the end-to-end rate adaptation performed by the sender, which achieves the desired partial reliability semantics.

As a result of the feedback aggregation, the rate feedback received by the sender contains the numbers of the packets received by the receivers connected via the slowest path and the fastest path in the multicast group, and the current sending rate of the immediate downstream MHPF server. Note that the feedback is generated periodically and the sender will see only one feedback per epoch, irrespective of the multicast group size.

When the sender receives the feedback, it performs LIMD rate control for both ρ_S and ρ_R according to \max_recv and \min_recv values, respectively. Once the new rates have been calculated, the sender controls the sending rate of the heterogeneous data stream using a *dual token bucket* mechanism described as follows.

Briefly, the sender maintains two token buckets: one for R-tokens generated at rate ρ_R , and the other for A-tokens generated at ρ_A . When the sender transmits a reliable packet it needs to acquire both R-token and A-token. On the other hand, when it transmits an unreliable packet, it only requires an A-token. It is straightforward that this dual token bucket mechanism upper-bounds the aggregate data rate by ρ_A , and the reliable data rate by ρ_R . In addition, it preserves the sequence of the interleaved reliable and unreliable packets. For example, even if there are A-tokens in the bucket, if the preceding reliable packet is waiting for an R-token, the unreliable packet behind the reliable packet must wait also.

C. Priority-based Packet Dropping

In Section III-B, we have identified that priority-based packet drop mechanism is one of the major components to achieve the goals of the MHPF service model. Essentially the priority dropping mechanism ensures that when the incoming data rate to the packet queue is greater than the outgoing data rate, only the highest priority data packets that are sustainable by the outgoing rate get forwarded by preferentially dropping low priority packets. For simplicity, we assume that each tunnel source maintains separate priority-drop queues for individual sessions. We will relax this assumption in the next section.

```

1  Variables for session i
2   $b_i$  // the number of packets in the buffer for session i
3   $B_i$  // the buffer bound for the session i

4  For incoming packet p
5   $pri \leftarrow p.priority$ 
6  if ( $b_i = B_i$ ) // if the queue is full, then perform priority dropping
7     $p' \leftarrow \text{find\_lowest\_priority\_packet}(i)$ 
8    if ( $p' < pri$ )
9      drop ( $p'$ ), then enqueue ( $p$ )
10   else
11     drop ( $p$ )
12   else
13     enqueue( $p$ )

```

Fig. 6. Pseudo code of the priority drop algorithm

The algorithm for priority dropping is quite straightforward (Figure 6). When the queue receives a new packet p , if the queue is full, then it searches for the lowest priority packet already in the queue. Then it compares the priority of the selected packet, say p' , with the new packet p . If the priority of p is higher than that of p' , then p' is dropped from the queue and p is enqueued. Otherwise, p is dropped.

Priority dropping involves fairly simple operations, and we typically anticipate only 4 priority levels in practice. However, the management of the per-session priority drop queue obviously incurs excessive overhead to MHPF servers as well as the per-session rate control. Hence we need to implement a smart session aggregation mechanism in order to alleviate the computational complexity at the MHPF servers.

D. Practical Issues

In this paper, we have thus far described all the mechanisms with respect to a single multicast session. Although we need to maintain state information and perform feedback aggregation on per-session basis, it will become impractical for MHPF servers to provide rate adaptation and priority dropping for each multicast session as the number of on-going multicast sessions increases. We now briefly outline a computationally simple approximation that allows the tunnel sources of MHPF servers to rate control and perform priority-drop for each *tunnel* rather than for each *session*.

The basic idea of session aggregation is the following: when there are multiple on-going sessions on a single tunnel, the tunnel source multiplexes them in a single output packet queue. In this case, the transmission rate on the tunnel is set to the sum of the data rates of all the sessions sharing the queue, i.e. $\rho_{tunnel} = \sum_i \rho_i, \forall i \in \text{active session}$. Likewise, the buffer bound of the queue is set to the sum of the buffer bounds of all the sessions, i.e. $B_{tunnel} = \sum_i B_i, \forall i \in \text{active sessions}$. We then adopt the *dynamic queue management* mechanism [19], which provides loose bandwidth assurances to each flow without performing per-flow weighted fair queueing.

We now briefly outline the dynamic queue management mechanism. Each flow has a buffer bound in a shared queue. A flow cannot enqueue a packet into the shared queue if both of the following conditions fail: (a) the aggregate queue size has exceeded the aggregated bound, i.e. $\sum_i b_i = B_{tunnel}$, and (b) the flow's queue size has exceeded its own buffer bound, i.e. $b_i = B_i$. Condition (a) allows for multiplexing the shared queue while condition (b) provides a minimum buffer allocation in the shared queue to each flow. It has been shown this simple multiplexing mechanism enables both efficient multiplexing with minimal state management while providing long-term bandwidth assurance for each flow. In our context, we approximate the rate adaptation mechanism simply by changing the buffer bound for the flow in the shared FIFO priority-dropping queue, and updating the aggregate sending rate on the tunnel.

Now we present the simulation-based performance results of MHPF to illustrate how our design achieves the desired service model.

V. PERFORMANCE RESULTS

We have instantiated the MHPF architecture in a laboratory testbed. The HPF protocol and the MHPF server functionality are implemented in the Linux 2.0.x kernel (Figure 7).

In addition to the implementation, we have been testing the performance of MHPF in comparison with other multicast protocols using *ns* simulator. While the testbed environment allows us to evaluate the protocol in a real network, the network configuration is limited in size. On the other hand, the simulation environment allows us to compare the performance of MHPF with other multicast protocols in a more controlled environment with various network configurations. In all our simulations, we have co-located MHPF servers with multicast routers for simplicity of presentation. In the rest of the section, we used the term “multicast router” and “MHPF server” interchangeably.

In this section, we present a set of simulation-based test results of MHPF in comparison with RLM (receiver-driven layered multicast). We chose RLM as a reference system since it is the most well-known layered multicasting ap-

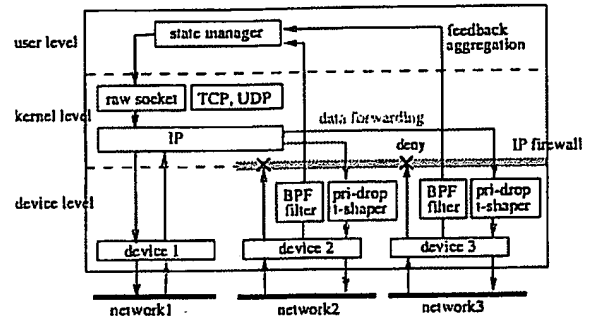


Fig. 7. The structure of the MHPF server implementation: on the forward path, packets are enqueued in the priority-drop queue of the custom traffic shaper associated with the outgoing tunnel. On the reverse path, the feedback is captured by BSD packet filter and pushed up to the state manager which performs feedback aggregation and rate adaptation. The original feedback is denied by firewalls and the aggregated feedback is forwarded via raw socket.

proach to date. Performance comparison with the later enhancements to RLM [8], [9],[10] is an on-going work.

We first consider a simple network topology shown in Figure 8. The available bandwidth is annotated on each link, and the one-way link latency was set to 10 msec for all the links. The operation parameters of RLM were adopted from [3]. We used 2.2 Mbps synthetic heterogeneous CBR flow with three priority levels with the ratio of high:medium:low at 1:2:3. In case of RLM, the sender transmits three individual CBR flows with 367 Kbps, 733 Kbps, and 1.1 Mbps for high, medium, low priorities. We measured the performance of each protocol for 100 seconds.

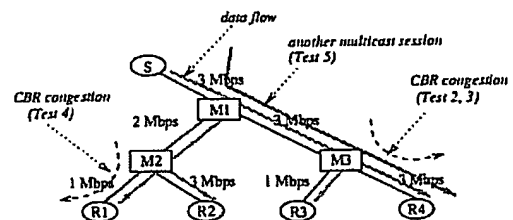


Fig. 8. A simple two-level topology

We first tested three cases: (a) steady state performance with no congestion, (b) adaptation to a long-term congestion, and (c) adaptation to a series of short-term congestions. We then present the effect of partial reliability, the session aggregation, and the performance in a larger network in the following tests. We omit other results, e.g. multiple congestion case, coexistence with TCP, due to the space constraints. Table I summarizes the results of the first five tests.

- *Test 1. Steady State Performance* In the first test, we compare the adaptation of each protocol to the heterogeneity of the network. The rate adaptation implementation of MHPF in *ns* closely simulates the behavior of fluid model,

		MHPF (Mbps)				RLM (Mbps)				MHPF / RLM (%)			
		R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4
Test 1 steady state	high	0.365	0.366	0.366	0.366	0.343	0.366	0.338	0.366	106	100	108	100
	mid	0.398	0.731	0.393	0.724	0.487	0.685	0.497	0.666	81.7	107	79.1	146
	low	0.001	0.428	0.002	0.971	0.005	0.005	0.005	0.784	20	8.6e3	40	123
	total	0.764	1.524	0.760	2.061	0.834	1.056	0.839	1.817	91.6	144	90.6	113
Test 2 long-term congestion	high	0.366	0.366	0.365	0.366	0.342	0.366	0.336	0.366	107	100	109	100
	mid	0.397	0.731	0.381	0.681	0.489	0.685	0.499	0.666	81.2	107	76.4	102
	low	0.001	0.426	0.004	0.827	0.004	0.005	0.004	0.016	25	8.5e3	100	5.2e3
	total	0.763	1.522	0.750	1.873	0.835	1.056	0.839	1.048	91.4	144	89.4	179
Test 3 short-term congestions	high	0.365	0.366	0.365	0.366	0.369	0.370	0.341	0.369	98.9	98.9	107	99.2
	mid	0.398	0.729	0.385	0.696	0.024	0.664	0.490	0.683	1.7e3	108	78.6	102
	low	0.001	0.421	0.004	0.888	0.002	0.008	0.005	0.046	12.5	5.2e3	80	1.9e3
	total	0.764	1.516	0.754	1.950	0.396	1.042	0.836	1.099	192	145	90.2	177
Test 4 partial reliability	high	0.337	0.344	0.344	0.344	-	-	-	-	-	-	-	-
	mid	0.380	0.680	0.365	0.674	-	-	-	-	-	-	-	-
	low	0.001	0.397	0.002	0.900	-	-	-	-	-	-	-	-
	total	0.717	1.422	0.710	1.918	-	-	-	-	-	-	-	-
Test 5 session aggregation	high	0.361	0.364	0.360	0.363	-	-	-	-	-	-	-	-
	mid	0.368	0.727	0.369	0.719	-	-	-	-	-	-	-	-
	low	0.006	0.404	0.006	0.307	-	-	-	-	-	-	-	-
	total	0.734	1.494	0.736	1.389	-	-	-	-	-	-	-	-

TABLE I

Effective throughput performance of MHPF and RLM at each receiver: The first four columns show the performance of MHPF, the next four columns show that of RLM, and the last four columns show the relative performance of MHPF over RLM.

thus we observe its LIMD-based rate adaptation achieves about 75 % of the path bandwidth for all the receivers. In practice, due to the buffering in network routers, we expect the bandwidth utilization of MHPF to be higher than 75 %. In case of RLM, we observed variable performance: more than 80 % bandwidth utilization for R_1 and R_3 , but only around 50 % for R_2 . Overall, we find both protocols achieve the goal of preferential delivery of high priority packets.

- **Test 2. Adaptation to Long-term Congestion** For this test, we instantiated 1 Mbps CBR flow on the link $M_3 \rightarrow R_4$ during 30 – 50 second period. In ideal case, we expect that only R_4 sees a temporary performance degradation, which is true for both protocols. However, there is a difference in the degree of degradation. In MHPF case, the throughput decrease is only around 9 % whereas the decrease of RLM is more than 42 % compared with Test 1.

- **Test 3. Series of Short-term Congestions** Now we compare the performance when there is a series of short-term congestions. On the link $M_3 \rightarrow R_4$, we instantiated 2 Mbps CBR flows during 30 – 31, 40 – 41, 50 – 51, and 60 – 61 second time windows. As in the previous case, we expect only R_4 sees performance degradation. This holds for MHPF. However, in RLM, there is performance degradation on R_1 , but we do not have explanation on this phenomenon. Overall, the performance degradation of R_4 in MHPF case is marginal (around 5 %) compared with Test 1. However, with RLM, R_4 sees around 40 % decrease even with short-term congestions.

Essentially the performance of MHPF in this case is in between that of Test 1 and Test 2, whereas the performance of RLM degrades to that of Test 2 (the long-term congest-

tion case). The poor performance of RLM in Tests 2 and 3 may be due to the coarse time grain rate adaptation based on join/leave operations and the exponential timer backoff after failed join experiments, which imposes potentially a long delay before rejoining the layer after the congestion has cleared out. From this experiment, we can deduce that the performance of RLM will become aggravated with the increase of the level of network dynamics.

- **Test 4. Partial Reliability** In this test, we show the impact of the partial reliability semantics (with receiver synchronization) of MHPF. According to the MHPF service model, if the slowest receiver in the multicast group is temporarily slower than the reliable data rate, then the entire multicast session will slow down in order to ensure the delivery of the reliable data to the slowest receiver. We set the high priority packet to be reliable and start a congestion on link $M_2 \rightarrow R_1$, which reduces the bandwidth by 0.7 Mbps for 5 seconds (and causes a small number of high priority packet losses). As a result, we observe not only R_1 but also all the other receivers experience the throughput degradation. However, note that MHPF is capable of terminating connection to excessively slow receivers in order to maintain the desired session progression of the group.

- **Test 5. Effect of Aggregation** We now briefly illustrate the effect of the aggregation using a simple scenario with two multicast sessions sharing the same multicast tunnel. As shown in Figure 8, we instantiated another multicast session going through M_1 and M_3 with the influx rate of 5 Mbps at M_1 . We assigned the same buffer bound to both multicast sessions, thus expect to observe approximately the same throughput for both sessions. The effective throughput of the original multicast session was mea-

	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8
MHPF	0.76	1.12	1.41	1.53	0.77	1.48	1.78	2.08
RLM	0.61	1.00	1.05	1.06	0.72	1.04	1.78	1.71
M/R (%)	126	112	136	144	106	142	100	122

TABLE II
Effective throughput results for Test 6

sured as 1.389 Mbps, and the new session was 1.289 Mbps. In other words, the approximation achieved by the session aggregation mechanism is quite effective (less than 4 % difference in the overall throughput) at least in this simple configuration. We have more results illustrating the effectiveness of dynamic queue management mechanism but we do not present them due to the space constraints.

• *Test 6. Performance in a Larger Network Topology*

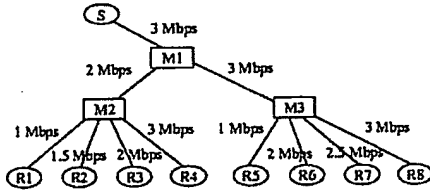


Fig. 9. An eight receiver configuration

In this test, we briefly present the performance of MHPF and RLM in a larger network topology shown in Figure 9. The effective data rate (Mbps) at each receiver in a steady state is summarized in table II. The major observation in this case is that the performance of RLM has decreased for all the receivers and the bandwidth utilizations are only 50 – 70 %. This must be resulting from the uncoordinated join/leave operations of the individual receivers since now there are more receivers sharing the same multicast router. On the other hand, we observe that the performance of MHPF is essentially the same compared with the previous test cases. On the average, MHPF still achieves around 70 – 75 % of the available bandwidth.

We tested out the same test in a larger multicast group of 16 receivers organized in a 4-level binary tree, and observed essentially the same trend: *as the number of receiver increases the performance of RLM degrades whereas the performance of MHPF remains almost the same.*

There can be several reasons for the progressively poor performance of RLM with the increase of the number of receivers and the network dynamics: (a) coarse grain adaptation, (b) interference of join/leave operations of individual receivers, and (c) incorrect congestion information propagated and maintained by the shared learning mechanism. By design choice, MHPF does not suffer from any of these problems. In summary, we observed that MHPF better utilizes the network bandwidth, effectively adapts to the network dynamics, and scales better in terms of the

multicast group size compared to the state of the art layered multicast approach RLM.

VI. SUMMARY AND REVIEW OF TRADE-OFFS

In this paper, we have presented a new approach for supporting multicast communications for the multimedia applications in the Internet environment. Unlike most related work, wherein layering mechanism is popularly used, we have proposed a multicast architecture where the multimedia data stream is processed as a single heterogeneous data stream and appropriate *filtering* mechanisms in the MHPF servers ensure the reception of highest priority portion of the data stream that can be sustained on each path of the multicast tree.

Having presented the design and performance of MHPF in previous sections, we now discuss the trade-offs in adopting the MHPF approach as opposed to conventional layering.

• *Granularity of adaptation* There are two dimensions to the granularity of adaptation: *how long* does it take to react to the network dynamics, and *how fine* is the adaptation in terms of bandwidth. The strength of MHPF is that it is highly adaptive along both dimensions.

In layering, a receiver determines when to join or leave multicast layers based on congestion estimation. Estimating congestion takes time; furthermore, joining/leaving layers is accomplished through IGMP, which are not optimized for reducing latencies. Consequently, layering approaches take longer to recover from short-term bandwidth fluctuations. In MHPF, rate adaptation takes place through a backpressure mechanism that takes effect within one epoch. Thus, MHPF is able to quickly recover from short-term rate fluctuations. Furthermore, because of the priority dropping mechanism, higher priority packets are not affected during adaptation of lower priority components unlike layering.

Even more serious point is the granularity of the bandwidth of rate adaptation. In layering, the sender pre-determines the granularity of rate adaptation by the number of layers into which it decomposes its heterogeneous data stream. Of course, each layer requires an independent multicast address – thus the sender is constrained to keep the decomposition coarse grain. On the other hand, each tunnel in MHPF is abstracted as a token bucket with priority drop queue – the efflux rate of the leaky bucket can be adjusted in a fine grain. In summary, MHPF adapts rate at the granularity of a packet, while layering approaches adapt rate at the granularity of a layer.

• *Choosing which component of the heterogeneous stream to receive* In MHPF, the sender determines the relative priority levels between the different interleaved sub-streams of a heterogeneous data stream, and all receivers must ad-

here to this notion of priority. A perfect application for this scheme is a multi-resolution video stream, where lower frequency or DC components have higher priority than higher frequency detail. In this case, all receivers have the same notion of priority, since it makes no sense receiving a lower priority packet at the expense of losing a higher priority packet.

However, in a heterogeneous data stream that is composed of interleaved audio, video and text data sub-streams, some receivers may prefer to receive audio with higher priority over video while other receivers may prefer video over audio. In this case, it is the receiver which must determine the priority of packets within a stream. MHPF is not well suited for this scenario, while layering allows the receivers the flexibility to choose their own layers. In summary, in MHPF the sender imposes the priority among data packets uniformly for all receivers, while in layering each receiver has the ability to establish its own priority scheme independently. While the latter is certainly more flexible, we believe that the MHPF model is adequate and well-suited for multi-resolution coding.

• *Deployment and Complexity* We compare MHPF and layering approaches in terms of computational complexity in the network and the end hosts, state requirements in the network, and ease of deployment.

In terms of computational complexity, layering moves the complexity within the network down to the routing layer (joins/leave operations), and to the end host (decomposition/resynchronization of data stream). Most layering approaches do not address the issue of reliable delivery. In MHPF, the complexity is spread out among the components – sender, receivers, and MHPF servers. The sender and receivers participate in per-session tasks such as flow control, partial reliability, and interleaving heterogeneous sub-streams. The MHPF server performs three tasks – rate adaptation and priority drop for each tunnel, and feedback aggregation for each session.

The key issue is whether the improved service model of MHPF is worth the computational complexity. For an optimized implementation, feedback aggregation occurs once every epoch (200 msec in our configuration) and requires between 50 to 100 instructions depending on the control path; priority dropping is per-tunnel and requires between 4 to 8 additional instructions (over tail dropping) for a priority queue with 4 levels; and rate adaptation requires approximately 20 instructions per-epoch per-tunnel. As we can see, the computational complexity for each of the components is somewhat reasonable. The main source of overhead comes from maintaining a traffic shaper for each tunnel (though traffic shaping is now a part of the standard distribution of most kernels, including our deployment platform of Linux 2.0.x). We are working on a tunnel-level window based flow control scheme that eliminates the traf-

fic shaping requirement. In summary, the processing overhead of MHPF is reasonable, and the fact that MHPF can be implemented using per-tunnel rather than per-session processing (except for reliability, which is an orthogonal issue) enables MHPF to scale well.

In terms of deployment, layering has the immense advantage of not requiring any change to the existing multicast servers. On the other hand, MHPF requires installing upgraded and specialized multicast servers, and thus requires some infrastructural change for deployment. In the long run, we believe that it is worth the trade-off because of the improved service model.

In conclusion, the issue of MHPF versus layering is really an issue of *filtering* versus *layering*. MHPF promotes the idea of sending a single heterogeneous data stream, and filtering through the highest priority component of the heterogeneous stream along each path of the multicast tree so that each receiver receives the most important data at the rate that its path can sustain. Additionally, MHPF provides mechanisms for reliable delivery of packets so marked by the sender, and supports end-to-end rate control of the interleaved reliable and unreliable sub-streams. MHPF offers a superior service model and enhanced performance for multimedia multicast communications at the expense of requiring smarts within the network. Irrespective of the practical issues of deployment, we believe that exploring this approach as a viable technical alternative to layering and understanding the trade-offs between the two approaches is important, and preliminary evaluations seem to indicate that MHPF is able to provide improved service at acceptable overhead.

REFERENCES

- [1] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. *INTERNET-DRAFT draft-left-avi-rtp-new-02.ps*, November 1998.
- [2] S. Cheung, M. H. Ammar, and X. Li. On the Use of Destination Set Grouping to Improve Fairness in Multicast Video Distribution. *Proceedings of IEEE INFOCOM '96*, March 1996.
- [3] S. McCanne and V. Jacobson. Receiver-driven Layered Multicast. *Proceedings of ACM SIGCOMM '96*, August 1996.
- [4] S. Floyd, V. Jacobson, C-G Liu, S. McCanne, and L. Zhang. A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing. *IEEE/ACM Transactions on Networking*, November 1996.
- [5] X. R. Xu, A. C. Myers, H. Zhang, and R. Yavatkar. Resilient Multicast Support for Continuous-Media Applications. *Proceedings of NOSSDAV '97*, May 1997.
- [6] S. Casner, J. Lynn, P. Park, K. Schroder, and C. Topolcic. Experimental Internal Stream Protocol, version 2 (ST-II). ARPANET Working Group Requests for Comment, RFC-1190, October 1990.
- [7] L. Delgrossi, C. Halstrick, D. Heilmann, R. G. Herrtwich, O. Krone, J. Sandvoss, and C. Vogt. Media Scaling for Audiovisual Communication with the Heidelberg Transport System. *Proceedings of ACM Multimedia '93*, August 1993.
- [8] L. Vicisano, J. Crowcroft, and L. Rizzo. TCP-like Congestion Control for Layered Multicast Data Transfer. *Proceedings of IEEE INFOCOM '98*, March 1998.

- [9] L. Wu, R. Sharma, and B. Smith. ThinStreams: An Architecture for Multicasting Layered Video. *Proceedings of NOSSDAV '97*, May 1997.
- [10] X. Li, S. Paul, and M. H. Ammar. Layered Video Multicast with Retransmissions (LVMR): Evaluation of Hierarchical Rate Control. *Proceedings of IEEE INFOCOM '98*, March 1998.
- [11] R. Gopalakrishnan, J. Griffioen, G. Hjálmtýsson, C. J. Sreenan, and S. Wen. A Simple Loss Differentiation Approach to Layered Multicast. *Proceedings of IEEE INFOCOM 2000*, March 2000.
- [12] J. Li, S. Ha, and V. Bharghavan. Transport Layer Adaptation for Supporting Multimedia Flows in the Internet. *Proceedings of IEEE INFOCOM '99*, March 1999.
- [13] S. Kumar, P. Radoslavov, D. Thaler, C. Alaettinoglu, D. Estrin, and M. Handley. The MASC/BGMP Architecture for Inter-domain Multicast Routing. *Proceedings of ACM SIGCOMM '98*, September 1998.
- [14] J.-C. Bolot, T. Turtletti, and I. Wakeman. Scalable Feedback Control for Multicast Video Distribution in the Internet. *Proceedings of ACM SIGCOMM '94*, September 1994.
- [15] T. Jiang, M. H. Ammar, and E. W. Zegura. Inter-Receiver Fairness: A Novel Performance Measure for Multicast ABR Sessions. *Proceedings of ACM SIGMETRICS '98*, June 1998.
- [16] K.-W. Lee, S. Ha, and V. Bharghavan. IRMA: A New Reliable Multicast Architecture in the Internet. *Proceedings of IEEE INFOCOM '99*, March 1999.
- [17] D. Bertsekas and Robert Gallager. Data Networks (Second Edition). *Prentice Hall*, pp. 506-507, 1992.
- [18] P. P. Mishra and H. Kanakia. A Hop by Hop Rate-based Congestion Control Scheme. *Proceedings of ACM SIGCOMM '92*, September 1992.
- [19] S. Ha, K.-W. Lee, and V. Bharghavan. Performance Evaluation of Scheduling and Resource Reservation Algorithms in an Integrated Services Packet Network Environment. *Proceedings of the Third IEEE Symposium on Computers and Communications '98*, July 1998.

Issues in Model-Based Flow Control

Sridhar Ramesh and Injong Rhee
Department of Computer Science
North Carolina State University
Raleigh, NC 27695-7534

Abstract

This report examines potential stability problems associated with a model based approach to TCP-friendly flow control for non-TCP traffic. In specific, such an approach involves using a TCP-friendly formula that estimates the throughput of a TCP session with the same end-to-end traffic characteristics as the non-TCP connection under consideration. The inputs to this formula include the round trip time, the timeout value, and the packet loss fraction of the connection. This paper shows that estimating the loss fraction per transmitted packet highly depends on the current transmission rate of the connection as well as the actual loss fraction of the path. Thus the estimated loss fraction can contain errors which result in inaccurate estimation of the corresponding TCP throughput. This inaccuracy can push the transmission rate of the non-TCP connection away from the fair share of the bottleneck bandwidth on the end-to-end path, so that under steady state, the connection ends up receiving either over-allocation or under-allocation of bandwidth.

1 Introduction

Congestion control is an integral part of any best-effort Internet data transport protocol. It is widely accepted that the congestion avoidance mechanisms employed in TCP [1] have been one of the key contributors to the success of the Internet. A conforming TCP flow is expected to respond to congestion indication (e.g., packet loss) by drastically reducing its transmission rate and by slowly increasing its rate during steady state. This congestion control mechanism encourages the fair sharing of a congested link among multiple competing TCP flows. A data flow is said to be *TCP-friendly* if at steady state, it uses no more bandwidth than a conforming TCP connection running under comparable conditions.

Recently we have seen several efforts to develop a stochastic model of TCP congestion control that gives a simple analytical formula for the throughput of a TCP sender as a function of packet loss and round trip time (RTT) [2, 7, 3]. These efforts are propelled by the interests in using the formula for TCP-friendly flow control of a non-TCP flow such as UDP traffic [6, 8] and reliable multicast [5]. Typically, these flow control schemes work as follows. A receiver monitors packet loss rates and round trip delays, and using a TCP friendly formula, it estimates the throughput of a TCP connection running under the same operating conditions. The estimated throughput is sent as feedback to the sender. If the feedback throughput is less than or equal to the current transmission rate of the non-TCP flow, then the sender sets its rate to the feedback throughput. Otherwise, it increases its rate.

The stochastic model used by Padhye et al. [3] makes the TCP throughput estimation based on the following assumptions:

- When a packet is lost, all subsequent packets in the same RTT round are lost.
- The probability that a packet is lost in an RTT round, given that no previous packet in the same round is lost is independent of packet loss in earlier rounds. Call this probability l_{act} .

To measure l_{act} of a TCP flow under observation, Padhye et al.[3] counts the number of TCP loss indications (triple duplicate acknowledgements, and timeouts) over a certain period, and divides the result by the total number of packets transmitted by TCP over that duration. This is an approximation. Let l_{app} be the expected value of the resulting value. l_{app} is used as input to their formula to estimate the throughput of a TCP connection under observation.

When a non-TCP flow is regulated using the formula, it is not possible to compute l_{app} since the flow may use a different window size. Instead, Handley et al. [5] and Padhye et al.[4] estimate l_{act} on the non-TCP flow by dividing the number of *loss events* by the total number of packets transmitted. Loss events are registered as follows. The first packet loss is counted as a loss event. Following this, there is a back-off for the duration of an RTT during which no packet loss is counted. The next

packet loss after this back-off is counted as a loss event, followed by another RTT back-off, and so forth. Let the expected value of this be l_{est} .

Let B be the total bandwidth shared among n TCP sessions, and one non-TCP flow running on the same end to end path. The objective of TCP-friendly algorithms is to ensure that the rate μ of the non-TCP flow is maintained at the fair share given by $B_{fair} = \frac{B}{n+1}$, in steady-state. The known TCP-friendly algorithms achieve this by a feedback mechanism. Specifically, an estimate of the throughput on each of the n TCP sessions is fed back to the sender which then uses this to regulate its transmitting rate μ . Assuming that the TCP sessions share the residual bandwidth equally, the steady-state throughput of each TCP session is $\frac{B-\mu}{n}$. Ideally, this is the parameter which must be fed back to the sender. It can be seen that when $\mu > B_{fair}$, $\frac{B-\mu}{n} < B_{fair}$, and when $\mu < B_{fair}$, $\frac{B-\mu}{n} > B_{fair}$.

However, as discussed earlier, the TCP throughput estimate is calculated based on the loss fraction of packets on the non-TCP flow. This obviously means that errors in estimating the loss fraction could lead to errors in estimating TCP throughput. In the following sections, we show that this indeed happens, and could even result in unfair allocation of bandwidth. In particular, we show that:

1. When $\mu > \frac{1}{RTT}$ (i.e., one packet per RTT), $l_{est} < l_{act}$. As a result, by substituting l_{est} for l_{act} in any formula which estimates TCP throughput based on l_{act} , we obtain an erroneous estimate. This results in over-estimation of the throughput.
2. When $\mu < B_{fair}$, $l_{est} > l_{app}$. Thus, substituting l_{est} for l_{app} in a formula estimating TCP throughput based on l_{app} gives us an erroneous estimate. This results in over-estimation of the throughput.
3. When $\mu > B_{fair}$, we could have $l_{est} < l_{app}$ under certain conditions. Thus, substituting l_{est} for l_{app} in a formula estimating TCP throughput based on l_{app} gives us an erroneous estimate and under-estimation of TCP throughput.
4. The formula used to estimate TCP throughput, such as the one provided in [3], may itself have some inaccuracy.

5. Depending on the starting rate of μ , these errors introduced in TCP throughput estimation push μ further away from the fair share, rather than forcing it to converge to the fair share. Thus, at steady state, the non-TCP flow may end up receiving either over-allocation or under-allocation of bandwidth.

In Section 2, we briefly outline the TCP-friendly flow control algorithm proposed in [5], and the formula used in TCP throughput estimation. In Section 3, we describe the sources of error in TCP throughput estimation, and show how these errors result in unfair allocation of bandwidth by the flow control algorithm. Section 4 contains numerical examples which substantiate these findings.

2 Model-based flow control

The first TCP throughput estimation model was proposed by Floyd [2] and Ott et al.[7]. Their model gives the following formula:

$$B(l) = \frac{1.22s}{t_{RTT}\sqrt{l}} \quad (2.1)$$

where l is the loss fraction of the TCP packets, and t_{RTT} is the round trip delay of an end-to-end path where TCP runs.

l is defined as the probability that a packet is lost given that no previous packet in the same *round* is lost (we use l_{act} and l interchangeably). l is estimated by counting the number of *loss events* over a certain period and dividing the result by the number of packets sent.

Padiye et al. [3] proposed another model which gives a better approximation than the earlier one. The model uses the following equation¹:

$$B(l) = \frac{s}{t_{RTT}\sqrt{\frac{2M}{3}} + t_0 \min\left(1, 3\sqrt{\frac{3M}{8}}\right) l(1 + 32l^2)} \quad (2.2)$$

where t_0 is the timeout value.

¹This is only an approximation formula. The complete formula can be found in [3]

Let μ be the bit rate of transmission of packets over the non-TCP connection. Let B be the total bandwidth available between the source and a given destination. If there are n active TCP sessions between the source and the destination, the throughput of each, in steady state, is given by $\frac{B-\mu}{n}$.

A typical model-based flow control works as follows. Using one of the above formulae, and estimated values of l , t_{RTT} , and t_0 , the receiver of the non-TCP flow computes a value of $B(l)$ which is fed back to its sender. If $B(l) \leq \mu$, the sender sets its transmission rate to $B(l)$. Otherwise, it increases its rate by some amount. The receiver continues to report a new value of $B(l)$, and the sender makes the adjustment of its rate accordingly. The idea is that if the feedback throughput value is accurate enough, the transmission rate eventually converges to the fair share of bandwidth (i.e., $\frac{\mu}{n+1}$).

3 Errors in TCP Throughput Estimation and their Impact on Resource Allocation

In this section, we look at the different sources that cause errors in estimating the equivalent TCP throughput and show that their potential consequence is unfair allocation of resources to the non-TCP flow.

3.1 Error in Loss Fraction Estimation

Let l_{act} be the actual loss fraction, and let l_{est} be the expected value of the estimate measured on the non-TCP connection. Let l_{app} be the expected value of an estimate measured on one of the TCP connections. Then, we have

THEOREM 1

$$l_{est} > l_{app} \text{ if } \mu < \frac{B - \mu}{n}$$

For proof, we compare the packets sent over the non-TCP stream over the measurement period T , to that sent over a TCP session. Let T be equal to N round trip

times. Let W_i , $i = 1, 2, \dots, N$, be the number of packets sent in the i^{th} round over the TCP connection under observation. Therefore, the number of packets sent over the TCP session is given by $\sum_{i=1}^N W_i$. The expected number of loss events is given by :

$$Loss_{TCP} = \sum_{i=1}^N \sum_{j=1}^{W_i} (1 - l_{act})^{j-1} l_{act}$$

Therefore, we have:

$$l_{app} = \frac{\sum_{i=1}^N \sum_{j=1}^{W_i} (1 - l_{act})^{j-1} l_{act}}{\sum_{i=1}^N W_i} \quad (3.3)$$

In the same duration, μT packets are transmitted over the non-TCP session. The expected number of loss events observed here is given by:

$$Loss_{Mcast} = \sum_{i=1}^N \sum_{j=1}^{T\mu/N} (1 - l_{act})^{j-1} l_{act}$$

Thus, we have:

$$l_{act} = \frac{\sum_{i=1}^N \sum_{j=1}^{T\mu/N} (1 - l_{act})^{j-1} l_{act}}{T\mu} = \frac{N \sum_{j=1}^{T\mu/N} (1 - l_{act})^{j-1} l_{act}}{T\mu} \quad (3.4)$$

Since $\mu < \frac{B-\mu}{n}$, and TCP is in steady state (congestion avoidance mode), there are more packets transmitted on any TCP connection, than on the non-TCP connection. Hence, $\sum_{i=1}^N W_i > T\mu$.

LEMMA 1 For any q such that $0 < q < 1$, $F_m(q) = \frac{1+q+q^2+\dots+q^{m-1}}{m}$ is a monotonically decreasing function m .

PROOF : $F_{m+1}(q) = \frac{1+q+q^2+\dots+q^{m-1}+q^m}{m+1}$, $q < 1$.

Hence, $q^m < q^{m-1} < q^{m-2} < \dots < q^2 < q < 1$.

$F_{m+1}(q)$ can be rewritten as $\frac{1+q^m/m+q+q^m/m+q^2+q^m/m+\dots+q^{m-1}+q^{m-1}/m}{m+1}$.

Thus, we have:

$$F_{m+1}(q) < \frac{1 + 1/m + q + q/m + \dots + q^{m-1} + q^{m-1}/m}{m+1} = \frac{1 + q + \dots + q^{m-1}}{m} = F_m(q)$$

LEMMA 2 If $G_m(q) = F_m(q) \times m = 1 + q + q^2 + \dots + q^{m-1}$, and $M = \frac{\sum_{i=1}^N m_i}{N}$, then

$$\sum_{i=1}^N G_{m_i}(q) \leq N \times G_M(q)$$

PROOF : Consider the sequence $\{q^{m_1}, q^{m_2}, \dots, q^{m_N}\}$. Its arithmetic mean is given by $AM(q) = \frac{\sum_{i=1}^N q^{m_i}}{N}$. Its geometric mean is $GM(q) = q^M$, where $M = \frac{\sum_{i=1}^N m_i}{N}$. $AM(q) \geq GM(q)$. Also, $0 < q < 1$, so $(1 - q) > 0$. Therefore, $\frac{N - AM(q)}{1 - q} \leq \frac{N - GM(q)}{1 - q}$. But $\frac{N - AM(q)}{1 - q} = \sum_{i=1}^N G_{m_i}(q)$, and $\frac{N - GM(q)}{1 - q} = N \times G_M(q)$.

We have:

$$l_{app} = \frac{\sum_{i=1}^N \sum_{j=1}^{W_i} (1 - l_{act})^{j-1} l_{act}}{\sum_{i=1}^N W_i} = \frac{\sum_{i=1}^N G_{W_i}(1 - l_{act}) \times l_{act}}{\sum_{i=1}^N W_i}$$

Therefore, from Lemma 2, it is seen that:

$$l_{app} \leq \frac{N \times G_{EW}(1 - l_{act}) \times l_{act}}{N \times EW} \quad \text{where } EW = \frac{\sum_{i=1}^N W_i}{N} \quad (3.5)$$

But $N \times EW = \sum_{i=1}^N W_i > \mu T$. Hence, from Lemma 1, we have:

$$\frac{N \times G_{EW}(1 - l_{act}) \times l_{act}}{N \times EW} < \frac{N \times G_{(\frac{T\mu}{N})}(1 - l_{act}) \times l_{act}}{T\mu} \quad (3.6)$$

Recognize that the RHS in (3.6) is l_{est} . Thus, combining (3.6) and (3.5) gives us the proof of Theorem 1.

THEOREM 2

$$l_{est} < l_{app} \quad \text{if } \mu > \frac{B - \mu}{n}, \quad \text{and } \frac{T\mu}{N} > W_i, \quad \text{for } 1 \leq i \leq N$$

From Lemma 1, it follows that $\frac{G_m(q)}{m} < \frac{G_k(q)}{k}$, for $k < m$. Therefore, $k \times G_m(q) < m \times G_k(q)$. Observe that l_{est} may be rewritten as :

$$l_{est} = \frac{\sum_{i=1}^N l_{act} \times G_{(T\mu/N)}(1 - l_{act})}{T\mu} = l_{act} \times N \times \frac{G_{(T\mu/N)}(1 - l_{act})}{T\mu}$$

Therefore, $l_{est} \times \frac{T\mu}{N} = l_{act} \times G_{(\frac{T\mu}{N})}(1 - l_{act})$. But $\frac{T\mu}{N} > W_i$, for $1 \leq i \leq N$.

Hence, for $1 \leq i \leq N$, we have:

$$G_{(\frac{T\mu}{N})}(1 - l_{act}) \times W_i < G_{W_i}(1 - l_{act}) \times \frac{T\mu}{N}$$

Taking the summation over $1 \leq i \leq N$, we get:

$$\sum_{i=1}^N G_{(\frac{T\mu}{N})}(1 - l_{act}) \times W_i \times l_{act} < \sum_{i=1}^N G_{W_i}(1 - l_{act}) \times \frac{T\mu}{N} \times l_{act} \quad (3.7)$$

Simplifying both sides, we get:

$$l_{est} \times \frac{T\mu}{N} \times \sum_{i=1}^N W_i < l_{app} \times \frac{T\mu}{N} \times \sum_{i=1}^N W_i$$

Or, $l_{est} < l_{app}$.

THEOREM 3 When $\mu > \frac{1}{RTT}$, $l_{est} < l_{act}$.

For proof, we revisit Lemma 1. Observe that:

$$l_{est} = \frac{\sum_{i=1}^N l_{act} \times G_{(T\mu/N)}(1 - l_{act})}{T\mu} = l_{act} \times \frac{G_{(T\mu/N)}(1 - l_{act})}{T\mu/N} = l_{act} \times F_{(T\mu/N)}(1 - l_{act})$$

But $\frac{T}{N} = RTT$, and $\frac{T\mu}{N} > 1$. Therefore, from Lemma 1, it follows that

$$F_{(T\mu/N)}(1 - l_{act}) < F_1(1 - l_{act}) = 1$$

Hence, $l_{est} < l_{act}$.

3.2 Impact of Errors in Loss Fraction on TCP Throughput Estimation

First, consider the case where TCP throughput is estimated as a function of l_{act} , such as (2.2). From (2.2), it can be seen that the estimate $B(l)$ is a decreasing function of l . Thus, when $l_{est} < l_{act}$, the estimate $B(l_{est})$ is likely to be larger than the actual value of $\frac{B-\mu}{n}$ which is the correct feedback parameter. In the extreme case, we may have the following situation:

$$B_{fair} < \mu < B_{fcd} = B(l_{est})$$

Consider the case where the non-TCP transmission rate μ is given by $\frac{B}{n+1} + n\Delta B$, for some $\Delta B > 0$. Now, suppose we are given a function $B(l)$ which correctly estimates the throughput on a TCP connection given the value of the loss fraction of packets, l . Let $\beta(l) = -\frac{dB}{dl} > 0$, for all l such that $0 < l < 1$. For the value of μ considered, $B(l) = \frac{B-\mu}{n}$, since we have assumed that $B(l)$ correctly estimates TCP throughput. Hence,

$$B(l) = \frac{B}{n+1} - \Delta B$$

Thus, when the value of μ is greater than the *fair share*, $\frac{B}{n+1}$, the feedback parameter is less than the fair share. This facilitates reduction of the transmission rate on the non-TCP connection to preserve bandwidth fairness.

However, when the feedback parameter is calculated based on l_{est} , the estimate of l rather than the actual value l_{act} , there is no such guarantee. As shown earlier,

$$l_{est} = \frac{\sum_{i=1}^N \sum_{j=1}^{T\mu/N} (1 - l_{act})^{j-1} l_{act}}{T\mu}$$

where T is the period of observation and N is the number of round trip times in T . Therefore, $l_{est} = l_{act} - \Delta l < l_{act}$. The feedback parameter $B(l_{est}) = B(l_{act} - \Delta l) \approx \frac{B}{n+1} - \Delta B + \beta(l_{act})\Delta l$. If $\Delta B < \frac{\beta(l_{act})\Delta l}{n+1}$, then we have:

$$B(l_{est}) \approx \frac{B}{n+1} - \Delta B + \beta(l_{act})\Delta l > \frac{B}{n+1} + n\Delta B = \mu$$

Next, we consider the case where TCP throughput is calculated using a formula based on l_{app} . Assuming we are provided a formula $B(\cdot)$ that accurately estimates TCP throughput, we should have:

$$B(l_{app}) = \frac{B - \mu}{n}$$

Let $\mu = \frac{B}{n+1} + n\Delta B$. But since we use l_{est} instead of l_{app} , the feedback parameter is given by:

$$B(l_{est}) \approx \frac{B}{n+1} - \Delta B + \beta(l_{app})(l_{app} - l_{est})$$

Thus, the following problems could be encountered:

1. When $\mu < \frac{B-\mu}{n}$, $l_{est} > l_{app}$. There could be a situation where $B(l_{est}) \leq \mu < B_{fair}$.
2. When $\mu > \frac{B-\mu}{n}$, and $l_{est} < l_{app}$, there could be a situation where $B(l_{est}) \geq \mu > B_{fair}$.

3.3 Errors in TCP Throughput Model

In the earlier section, we considered the error due to inaccurate estimation of the loss fraction, assuming that the throughput model to be perfect. However, it is seen that

the throughput estimation based on (2.2) also introduces some error (see [3]). From the tables provided in [3], it was observed that the relative error approached 100% in certain extreme cases. In this section, we show that this error can add on to the error in loss fraction estimation to result in erroneous feedback to the non-TCP sender.

Suppose the model estimates TCP's throughput to be $B_{mod}(l) = B_{act} + B_c$, where B_{act} is the actual TCP throughput.

Consider a link between two nodes with a minimum round trip time of T_1 seconds. Let $N_1 = \lfloor B_{act} \times T_1 \rfloor$. Let $l_1 = l - \Delta l = l \times \frac{1-(1-l)^{N_1}}{N_1}$. It is easily seen that $\Delta l > 0$ if $N_1 > 1$. Let the bandwidth of the link under consideration be $B_{tot} = nB_{act} + \frac{B_{mod}(l) + B_{mod}(l - \Delta l)}{2}$. Now, suppose we have a flow of rate $\mu = \frac{B_{mod}(l) + B_{mod}(l - \Delta l)}{2}$, sharing this link with n , $n \geq 1$, TCP connections. The throughput of each TCP session is $\frac{B_{tot} - \mu}{n} = B_{act}$. Using (2.2) however, we get different results. If l is estimated correctly, the TCP bandwidth is estimated as:

$$B_{mod}(l) = B_{act} + B_c$$

Since $\Delta l > 0$, and $B_{mod}(l)$ is a decreasing function of l , $B_{mod}(l - \Delta l) > B_{mod}(l)$. So,

$$\mu - B_{mod}(l) = \frac{B_{mod}(l - \Delta l) - B_{mod}(l)}{2} > 0$$

In other words, $\mu > B_{act} + B_c$.

But l is not correctly estimated. The estimate of l is given by:

$$l_{est} = \frac{1 - (1 - l)^{\mu \times RTT}}{\mu \times RTT}$$

where RTT is the mean round-trip time. Obviously, $RTT \geq T_1$. Besides, $\mu > B_{act} + B_c$. Hence, from Lemma 1, it can be seen that $l_{est} < l_1 < l$. We can write $l_{est} = l - Dl$ where $Dl > \Delta l$.

The estimate of the TCP bandwidth, using (2.2), is given by:

$$B_{mod}(l_{est}) = B_{mod}(l - Dl)$$

As mentioned earlier, $B_{mod}(l)$ is a decreasing function. Therefore,

$$B_{mod}(l_{est}) = B_{mod}(l - Dl) > B_{mod}(l - \Delta l)$$

Also,

$$B_{mod}(l_{est}) - \mu > B_{mod}(l - \Delta l) - \mu = \frac{B_{mod}(l - \Delta l) + B_{mod}(l)}{2} > 0$$

The fair share is given by:

$$B_{fair} = \frac{B_{tot}}{n+1} = B_{act} + \frac{B_{mod}(l - \Delta l) - B_{mod}(l)}{2n+2} < B_{act} + B_c + \frac{B_{mod}(l - \Delta l) - B_{mod}(l)}{2} = \mu$$

Thus, we have a situation where $B_{mod}(l_{est}) > \mu > B_{mod}(l) > B_{fair}$, i.e., one where the error in estimating l , coupled with the error introduced by the formula (2.2) results in erroneous throughput estimation. In the next section, we show how this may even result in unfair allocation of resources.

3.4 Non-Convergence to Fair-Share

In the earlier sections, we have shown that the errors in estimating the loss fraction and errors in the TCP throughput formula could result in the following situations:

$$(i) B_{feed} > \mu > B_{fair} \quad \text{OR} \quad (ii) B_{feed} < \mu < B_{fair}$$

where μ is the rate of the non-TCP flow, B_{fair} is the fair share, and $B_{feed} = B(l_{est})$ is the feedback parameter.

In this section, we show that when $B_{feed} > (\text{resp } <) \mu > (\text{resp } <) B_{fair}$, the non-TCP rate never converges to the fair share in steady state. In addition, the long-term average throughput of the non-TCP flow does not converge to the fair share either.

We prove this for the case where $B_{feed} > \mu > B_{fair}$, and the proof for the case $B_{feed} < \mu < B_{fair}$ is similar. We first observe that $B_{feed} = B(l_{est})$, where l_{est} is itself a function of μ . Therefore, we can write $B_{feed} = B^\Omega(\mu)$. Assuming $B(l_{est})$ is a continuous function, and l_{est} is a continuous function of μ , $B^\Omega(\mu)$ is also continuous. We know that $B^\Omega(\mu) > \mu$. Hence, we have either:

$$B^\Omega(\nu) = \nu \quad \text{for some } \nu > \mu \tag{3.8}$$

Or:

$$B^\Omega(\nu) > \nu \quad \text{for all } \nu > \mu \tag{3.9}$$

If (3.8) is true, then the feedback parameter when the initial rate of the non-TCP flow is ν is given by $B^\Omega(\nu) = \nu$. Therefore, the rate as determined by the non-TCP flow control protocol converges to $\nu > \mu > B_{fair}$. The long-term average of the flow is also ν .

If (3.9) is true, when the initial rate ν_0 is greater than μ , it can be seen that the feedback parameter $B^\Omega(\nu_0)$ is greater than ν_0 . Writing $\nu_1 = B^\Omega(\nu_0)$, and in general, $\nu_{i+1} = B^\Omega(\nu_i)$ for $i > 0$, we have:

$$B^\Omega(\nu_i) > \nu_i \geq \nu_0$$

In practice, $\nu_{i+1} = \min(B^\Omega(\nu_i), B_{max})$, but since $\nu_i \leq B_{max}$, we can still write:

$$\nu_{i+1} \geq \nu_i \geq \nu_0$$

This means $\nu_i > \mu > B_{fair}$, for all $i \geq 0$. The long-term average of the flow is given by:

$$\gamma_\infty = \lim_{N \rightarrow \infty} \frac{\sum_{i=0}^{N-1} \nu_i}{N} > \lim_{N \rightarrow \infty} \frac{\sum_{i=0}^{N-1} \mu}{N} = \mu > B_{fair}$$

4 Numerical Examples

We consider some numerical examples of cases where the flow control algorithm using formula based feedback results in unfair allocation of resources.

In the first example, the formula is assumed to accurately compute the TCP throughput when the loss rate is correctly estimated, and the error in TCP throughput estimation is due to erroneous estimation of the loss fraction. There are two TCP sessions sharing the bottleneck link which has a bandwidth B of 500 KB/s. The fair share is therefore 166.7 KB/s. We assume that the non-TCP flow has an initial rate μ of 190 KB/s. An ns simulation was accordingly set-up, with two TCP connections sharing a 4 Mb/s link with a constant bit-rate source sending packets at 190 KB/s. The observed loss fraction in the simulation is 0.0156. However, the loss fraction as estimated by dividing the number of loss events by the total number of packets transmitted is 0.0125. Using the same simulation set-up, it was determined that a loss fraction of 0.0125 corresponds to an actual transmitting rate μ of 120 KB/s by

the constant bit-rate source. Therefore, any correct formula estimates the TCP rate as $\frac{B-\mu}{n} = \frac{500-120}{2} = 190$. In other words, the feedback parameter B_{fed} is equal to μ though μ is significantly larger than the fair share. Since $\mu = 190$, this leaves less than 155 KB/s for either of the TCP connections. Thus, the bandwidth allocation to μ is at least 23% greater than that for each TCP connection.

In the second example, we illustrate the effects of estimating the TCP throughput based on the formula given in (2.2). There could be an error introduced by the formula, compounded by an error due to inaccurate estimation of the loss fraction. Among the measurements provided in [3], we consider the case where 100 serially initiated TCP connections were established for 100 second intervals between two hosts. An average throughput of 17.13 KB/s was observed per connection, with a loss fraction of 0.0078, mean RTT of 0.2501 seconds and time out period of 2.5127. The throughput estimation, as given by (2.2) was 33.4 KB/s. Now, suppose the bottleneck is a 496 KB/s (≈ 4 Mbps) link shared with 27 TCP sessions. The fair share is 17.75 KB/s, but when $\mu = 33.5$ KB/s, the TCP throughput is 17.13 KB/s. Assuming the TO and RTT parameters are the same as in the measurement described above, the approximate loss fraction l_{app} is .0078, and the formula in (2.2) estimates the TCP throughput as 33.4 KB/s. Assuming $l_{act} \approx l_{app}$, and applying the transformation for l_{est} given in (3.4), the estimate of the loss fraction on the non-TCP connection is .0076. Substituting this value for l in equation (2.2), $B_{fed} = B(l)$ is 34.01 KB/s which is higher than μ although μ itself is higher than the fair share. Thus, the resource allocation to μ is atleast about 89% higher than the fair share.

References

- [1] V. Jacobson, "Congestion Avoidance and Control," *Proceedings of ACM SIG-COMM'88*, August 1988.
- [2] S. Floyd, "Connections with Multiple Congested Gateways in Packet-Switched Networks Part 1: One-way Traffic," *Computer Communications Review*, vol. 21, no. 5, October 1991.

- [3] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modelling TCP Throughput: A simple model and its empirical validation," *Proceedings of SIGCOMM '98*, 1998.
- [4] J. Padhye, J. Kurose, D. Towsley, and R. Koodli, "An Empirical Study of Client Interactions with a Continuous-Media Courseware Server," *IEEE Internet Computing*, April 1999.
- [5] M. Handley, and S. Floyd, "Strawman Specification for TCP Friendly (Reliable) Multicast Congestion Control (TFMCC), *Reliable Multicast Research Group*, December 1998.
- [6] J. Mahdavi, and S. Floyd, "TCP-Friendly Unicast Rate-Based Flow Control," *Technical Note on End2end Mailing List*, January 1997.
- [7] T. Ott, J. Kemperman, and M. Mathis, "Window Size Behavior in TCP/IP with Constant Loss Probability," *DIMACS Workshop on Performance of Realtime Applications on the Internet*, Plainfield NJ, November 1996.
- [8] D. Sisalem, F. Emanuel, and H. Schulzrinne, "The Direct Adjustment Algorithm: ATCP-Friendly Adaptation Scheme," *Preprint*, Columbia University, 1998.

Heterogeneous multicast congestion control based on router packet filtering

Michael Luby*
luby@dfountain.com
Digital Fountain, Inc. and ICSI

Lorenzo Vicisano, Tony Speakman
{lorenzo,speakman}@cisco.com
Cisco Systems, Inc.

May 31, 1999

Abstract

We propose a new IP multicast congestion control protocol based on intelligent router packet filtering that allows data rate differentiation in presence of heterogeneous network conditions. A primary target application is reliable content distribution from a single server to a large population of independently acting receivers. The objective is to enable receivers with varying bandwidth availability to each receive a copy of the content at the maximum rate allowed by their fair usage of the network, independent of other receivers. For this application, our congestion control protocol is ideally combined with an FEC reliability protocol.

1 Introduction

We propose a new IP multicast congestion control protocol based on intelligent router packet filtering. The objective is to enable receivers with varying bandwidth connections to the server to each receive packets at their maximum fair rate independent of other receivers.

A primary target application is reliable content distribution from a single server to a large population of independently acting receivers. For this application, our congestion control protocol is ideally combined with a reliability protocol that ensures that any receiver that collects any subset of

*Supported in part by NSF operating grant CCR-9800452.

packets that are equal in aggregate length to the length of the content being distributed can reliably recover the content. We hereafter call such a reliability protocol an FEC protocol. As we describe in more detail in section 7, the combination of an FEC protocol and our new congestion control protocol allows receivers to join an ongoing session at any point during the session and reliably download content at the speed of their connection to the server independent of other receivers. For other applications, such as streaming multimedia content distribution, other reliability protocols are ideally matched with our congestion control protocol. We discuss one such application and reliability protocol briefly in section 8.

1.1 Congestion control protocol overview

One of the main challenges in heterogeneous rate and congestion control is regulating the packet rate in specific points of the network, so that from that point a reduced rate flow is transmitted downstream, given that a certain —possibly higher— rate is received from up-stream. The congestion control protocol presented here controls the rate that packets flow to each receiver with no involvement by either the source or the receivers. The routers perform all of the flow and congestion control, where the mechanism used to slow down packet flow rates along certain paths is router based packet filtering. An FEC protocol should be used in conjunction with this congestion control protocol for reliable content delivery in order to guarantee that each receiver can reconstruct the content from a minimal amount of packet reception independent of the overall filtering rate from the source to that receiver.

The congestion control protocol consists of four parts:

1. Link congestion measurement
2. Upstream congestion reports
3. Interface rate computation
4. Interface rate filtering

Link congestion measurement monitors the packet drop rate along each link in the multicast tree. This measurement is not particular to the IP multicast flow across the link, but measures the aggregate of all flows across the link, which could include competing TCP/IP traffic as well as other IP multicast flow traffic from other sessions. Upstream congestion reports

are used to report the congestion measurements appropriately up the IP multicast tree—from the receivers towards the multicast source—to the point of regulation. Interface rate computation uses the congestion reports to determine the rate at which packets should be allowed to flow along each outgoing router interface for the IP multicast session. A possible desirable property is that the computed IP multicast session rate achieves the fair amount of the bandwidth across the interface, but no more (fair to other flows across the interface, e.g., TCP/IP flows). The way congestion reports are translated into interface rates is not explicitly defined here, as it depends on the desired form of fairness. However possible formula-based ways of inferring these rates are mentioned. Finally, an interface rate filtering policy is used to ensure that the IP multicast packets for the session flow out of each outgoing interface at the computed rate for that interface. This policy is essentially a simple packet filtering.

2 Other congestion control solutions

A common approach to multicast congestion control is sender-based rate regulation [4]. This simplifies the task of regulation itself but precludes the possibility for rate differentiation, posing the problem of choosing what the single rate should be. The most common solution is having the source that throttles its outgoing packet rate down to the lowest among the bandwidth shares sustainable by a participating receiver or by a crossed link. This conservative approach assures that the multicast session does not use more than its fair share on any of the paths or links crossed. Thus, participating receivers that have higher available bandwidth paths to the server suffer at the expense of receivers that have lower available bandwidth paths.

Solutions that support rate differentiation on multiple end-to-end path include layered multicast [8, 15, 3]. These solutions have basically the same properties as the overall congestion control protocol proposed here, except that they perform coarser grain congestion control (in terms of possible achievable rates and reaction speeds to changes in network conditions) and they involve receivers in the congestion control instead of routers. This poses some inter-receiver coordination challenges but has the advantage that doesn't need router modifications. Another disadvantage with the layered solution is that it requires the use of multiple multicast group to implement a single session, which is not as efficient for source, receivers, and network resource utilization.

Store and forward solutions also address the heterogeneity issue by means

of caching incoming packets at specific nodes and retransmitting them downstream at lower rates. Compared to other approaches, these solutions utilize network bandwidth the best — packets are transmitted only once on each link— and the source does not have to keep transmitting packets once downstream nodes have a complete copy, and this property iterates to nodes further downstream. This solution however presents drawbacks and not completely solved problems. The major disadvantage is the presence of store-and-forward agents that needs to be managed and administrated (including space requirements and fast access to stored packets for retransmission). Store-and-forward agents also introduce the unresolved issue of dynamically placing/activating them in the points where their action is needed, unless their ubiquitous presence is advocated, in which case the resources needed, in term of storage and protocol-processing power, could be too large to make this a viable solution.

A direction of possible future research involves combining the solutions proposed here with caching solutions. Another possible direction for future research involves considering a solution where multiple sources are multicasting the same content, and receivers can hook into one or more of the streams emanating from the sources. The reason for considering this is the possibilities for additional reliability and increases in performance. Performance may improve due to better load balancing of the bandwidth throughout the network.

3 Interface rate filtering

This section describes how routers regulates an IP multicast flow on their outgoing interfaces, assuming that they receive appropriate congestion reports from downstream nodes, and that the congestion control algorithm they run turns these congestion reports into packet rate to use on the outgoing interfaces.

Each router is aware of the packet flow rate R from the source. This information is delivered to the routers in the IP multicast tree through SPMs [12] (session path messages). SPMs also have the function of starting the protocol machinery along the multicast tree¹. We assume that the upstream congestion reports have already been used to compute an interface filter rate f_i for each outgoing interface i . The value of f_i is the fraction of

¹SPMs can also be used to carry other information such as dynamically configurable session parameters, and can serve as means to discover protocol-capable upstream neighbors.

the source packet flow rate that should be allowed to flow through interface i , so that the throughput for the session through interface i is $f_i R$.

Suppose that there is an interface i' upstream of interface i that has a filter rate $f_{i'}$. A key goal is to ensure that if $f_{i'}$ is less than f_i then no additional filtering takes place at interface i , and if $f_{i'}$ is greater than f_i then the packet flow is filtered down from rate $f_{i'}$ to rate f_i at interface i by filtering out a fraction $f_{i'} - f_i$ of the original source packet flow.

The filtering mechanism we recommend relies on a sequence number included in each packet. The basic idea is to filter out packets based on their sequence number as follows. Suppose that a packet is received for possible transmission out interface i with sequence number $a_0 a_1 a_2 \dots a_{s-1} a_s$, where a_j is the j -th bit of the sequence number. Consider the binary number $b = 0.a_s a_{s-1} \dots a_2 a_1 a_0$, i.e., b is the sequence number written in reverse order expressed as a real number that is between 0 and 1. Then, the packet is only allowed to go out of interface i if $b \leq f_i$, and otherwise the packet is filtered out. It turns out that this simple scheme satisfies the key goal described above, and it also has the property that the flow rate out of the interface is as smooth as is possible assuming packets are not lost for other reasons and that sequence numbers are consecutive².

It is important when measuring drop rates on interfaces to not count packets that are intentionally filtered out, as these drops are not a consequence of congestion but a mean to perform flow regulation.

4 Upstream congestion reports

Assuming that estimates of the drop rate per outgoing interface are available at every router, this section describes how the appropriate congestion signal is reported from the routers near the receivers upstream towards the source to the nodes which perform flow regulation.

Two options seem viable to report congestion upstream. The most straightforward one is using some direct metric of congestion, such as the packet drop rate on outgoing interfaces. The other option is computing the data rate that a router is willing to receive —based on the rates which the router is allowed to transmit on its outgoing interfaces— and reporting this

²This turns out to be a consequence of some deep work in discrepancy theory by a mathematician by the name of Schmidt and others that is beyond the scope of this document. Schmidt won a Fields medal in part for proving that there is no smoother scheme, and this is the hard part. Computing the smoothness of this scheme is not that hard. It turns out that it is a lot smoother than would be obtained by doing random filtering.

upstream. In this case the rate computation is performed where congestion occurs.

Although the first solution seems the most obvious, the second is preferred as it allows more flexibility. With this, for example, it is possible to bias the aggressiveness of the flows in sharing the bandwidth according to topological considerations³.

The basic rule to collect/forward congestion reports is the following. For each outgoing interface i from router r , r computes a filter rate g_i based on the drop rate (congestion) on this interface —recall that a filter rate is a fraction of the packets transmitted by the source that needs to be filtered out to achieve the throughput established by the congestion control policy. For each outgoing interface i from router r , r has received a filter rate h_i from the downstream router along this interface for the IP multicast session (initially, h_i is set to 1). Then, the target filter rate f_i for the session along outgoing interface i is set to the minimum of g_i and h_i . When router r passes a filter rate to the upstream router for the session it sends the maximum of the f_i s over all interfaces i that carry the session. Router r also uses the value of f_i as a target to perform packet filtering for the session along each interface i .

The rationale behind the rule is that the rate of packet flow down an interface i should not exceed the value g_i that is computed based on its fair share. On the other hand, if this is not the limiting rate, then the packet flow rate should also not exceed the value h_i that the downstream router has indicated that this interface can handle. Finally, the report of the filter rate to the upstream router from router r should be set to the maximum of all the rates that all the interfaces out of r can handle.

Note that, once a new value for f_i is computed on interface i , this might either be used immediately as filter rate to apply to that interface, or it might determine the actual filter rate in some less direct way. As an example, the congestion control scheme used might impose that a rate increase in an interface is not applied immediately but gradually through a smoothing low-pass function.

The amount of memory needed to implement this scheme is $k + 1$ words at outgoing interface i if there are k multicast sessions that are currently sending packets through interface i —one word for the h_i of each session and one for g_i (global to all the sessions).

³A possible application for this is biasing the aggressiveness according to the distance from the root of the tree, so that flows which aggregate more potential receivers (on links closer to the root) can given higher priority. Priority can also be based on the number of downstream receivers, if this information is available.

4.1 Report periodicity

Each node (router) generates periodic reports upstream to refresh the state in its parent router and possibly in some of its direct ancestors. The inter-reports period determines the protocol responsiveness to changes in network load and the accuracy of the control system. It also affects, however, the protocol overhead in terms of control traffic. Possible options to determine this period are either using a protocol constant, tuning it through SPMs or making it a function of the speed of variation of the value being reported. This adaptive solution is preferred as it allows dynamic tuning of the periodicity, increasing the control traffic only if needed. The following heuristic is suggested to implement the adaptive scheme without incurring error accumulation: reports are sent when the previously sent report differ by more than δ from the current value (determined as the maximum of the f_i s). However, if no reports have been sent for t_{max} seconds, a report is sent in any case. The purpose is to refresh an upstream state that may be out of date due to lost reports or a number of too small to report changes in the filter rate.

The value of δ should be a small percentage of the actual value. It should be chosen also taking into account the method used to measure the interface congestion statistics (see section 5): it should be larger than the expected measurement error, so to prevent high report traffic due to measurement noise only. The value of t_{max} should be short enough to provide the degree of error resilience wanted, its lower limit being determined by how frequently the congestion statistics are updated.

To assess the overhead generated by control traffic (reports), consider both the case of a completely static situation (load values in links do not change) and the case where the load distribution across the network changes much slowly than the frequency of reports. In the static situation, where congestion conditions do not vary, each node generates periodic reports every t_{max} seconds, which cross one link only, as the updates received in each node do not change the local status. In the dynamic situation, consider the most general case of router that is root of n sub-trees. It will receive reports from all of them with a certain periodicity. Not all the reports however will affect the maximum f_i : If we assume that congestion distribution varies slower than the report periodicity, only one among the subtree reports and the local g_i s will affect the maximum f_i , hence the router will generate reports with the same periodicity as that of the dominant subtree or the one imposed by one of the local g_i s. If we apply this recursively, we can see that the periodicity is determined by one of the g_i s of some router, thus

the upper bound for the overall report traffic is given by the speed of load changes in that link. This is in turn bounded by the measurement process (see section 5).

5 Interface congestion measurement

Each router collects traffic statistics on outgoing interfaces and uses them to infer traffic load on links. A number of different observations could be made on link packet-queues to estimate link utilization, which include queue size and number of packets dropped. Using queue size —or other direct measurements of the link load— allows to perform congestion control with zero —or very small— steady loss rate. On the contrary, using loss rate implies that the stable operating point is reached only in presence of a certain loss rate in links, as there would not be feedback at all otherwise. In other words congestion control is performed by slightly overloading the network and using the observed loss rate to decide whether to increase or decrease the offered load.

Although TCP is indirectly based on both, the loss rate is one of the factors that mainly determines the long term data rate. This means that, in presence of competing TCP flows, any degree of fairness can only be reached with significant loss rate in the steady operating condition. In provision for this we base our congestion statistics on percentage of packets dropped on routers outgoing link queue.

Records on the number of packets forwarded and dropped already exist in routers, hence the only addition required to estimate the percentage of packets dropped is a function that compute a moving average of the ratio of packets dropped versus packets queued to be forwarded. A simple way to do this is using an exponentially weighted average, where the weighting factor is chose considering responsiveness requirement and measurement error issues.

6 Interface rate computation

The algorithm used to turn the congestion statistic into a target data rate and to make the current rate converge to the target rate determines the form of fairness achievable and the stability of the control system. Although covering this issues is beyond the scope of this document, we will discuss briefly the choice of the function that determines the long-term data rate from the congestion measurements.

The most natural usage of this architecture is implementing Max/Min fairness, where bandwidth is shared on a per-link basis. A simple way of implementing this kind of fairness is translating congestion statistics into link rates through a monotonically decreasing function. If all routers use the same function for all the flows, then Max/Min fairness is automatically achieved despite the actual shape of the function. The shape of the function however determines the stability of the system, determines its convergence speed and determines the steady loss rate in links as a function of the number of flows crossing them.

If the function chosen is one of the flavors of the TCP equivalence formula [7, 9], then an approximation of TCP-like fairness can be implemented. TCP-like fairness is generally extended to multicast through the following definition "a multicast session is TCP-fair if its rate on any of the end-to-end paths source-receiver is lower or equal to the rate that a TCP connection would achieve in that path". The long-term rate of a TCP connection depends on the cumulative loss rate of the path traversed and on its round-trip time (RTT). As we do not have any provision for estimating RTTs and for cumulating path loss rate —although these could be easily introduced in our architecture— we can only achieve some kind of approximation of TCP-fairness. The simplest approximation would be using single link loss rate and an average-TCP connection RTT, say \overline{RTT} , and the TCP-equivalence formula to compute the rate. This approximation would provide a form of fairness that can be stated like this: "the multicast session achieves in each link a rate never (much) larger than the rate a TCP connection crossing that link would achieve if that link was the dominant bottleneck in the TCP path and if the path RTT was \overline{RTT} ".

The shape of the function used to compute the rate can also be determined on a per-session basis and/or changed dynamically. A simple way of achieving this is using session path messages (SPMs) to carry the function parameters. This would allow easy tuning of several parameters to determine the congestion control behavior, such as aggressiveness and responsiveness.

Another interesting possibility is using slightly different functions in different nodes of the session tree. This would, for example, allow to bias the flow aggressiveness in sharing bandwidth, according to the distance from the source, the number of downstream receivers or the node fan-out. In this way multicast specific sharing policies (e.g. [5]) could be easily implemented.

7 Content Delivery

A primary target application of the new congestion control protocol is a reliable content delivery system from a single server to a large population of independently acting receivers. For this application, our congestion control protocol should be used with an FEC protocol. An FEC protocol or an approximation thereof can be constructed using either standard or more advanced FEC codes, e.g, see [11, 6, 3]. Recall that an ideal FEC protocol ensures that any receiver that collects any subset of sent packets that are equal in aggregate length to the length of the content being distributed can reliably recover the content. Thus, for an FEC protocol, complete recovery of the content depends only on the number of packets received, and the number required to recover is independent of packet loss patterns and when the receiver joined and left the session.

With our congestion control protocol, packets are intentionally filtered out to adjust the flow rate to each individual receiver down to their fair share of the bandwidth connection to the server. Because recovery of the content depends only on receiving the minimal possible number of packets independent of loss patterns when using an FEC protocol, this leads to the ideal behavior that each receiver recovers the content at their maximum fair rate independent of other receivers. This leads to a reliable content delivery system with the following desirable properties.

- Each receiver is able to receive packets at the maximum fair rate at each point in time independent of the other receivers, e.g. a receiver that has a more constrained bandwidth connection to the source should not slow down transmission to receivers with less constrained bandwidth connections.
- Each receiver is able to join and leave the session at its discretion without adversely affecting the source or the other receivers.
- The source is able to transmit packets at a single fixed rate to one multicast session. This is the maximum rate available to receivers.
- Receivers are able to receive packets without any feedback required for congestion or rate control.
- Each receiver is able to recover the content as soon they have received a number of packets that have aggregate length equal to that of the content, and this is independent of the other receivers.

8 Layered Video

Layered video is an application that is well-suited for our new congestion control protocol. The basic idea behind layered video is that there are several streams of video. Reception of the base layer stream provides a reasonable quality video playback experience, and as more and more enhancement layers are received the quality incrementally improves. These layers are ordered, in the sense that the first enhancement layer is of no use unless the base layer is received, the second enhancement layer is of no use unless both the base and the first enhancement layer are received, etc. Multicast schemes have been proposed that allow different receivers with heterogeneous connections to the server to receive a quality of video stream that is proportional to their connection (see e.g [13] or [14]).

We propose a reliability protocol that could be used in conjunction with the congestion control protocol already described for this application. Suppose that the full video stream is being transmitted at some rate R . Suppose that layer i of the stream comprises a fraction p_i of the raw stream. Let $c_0 = 0$ and in general let $c_i = \sum_{j \leq i} c_j$. Then, one naive way to send the stream in conjunction with the congestion control protocol is to let the i -th stream be placed into packets with sequence number j such that when j is written in reverse binary as a number between 0 and 1 its value is between c_{i-1} and c_i . This ensures that when packets are intentionally filtered out that the enhancement layers are filtered out first in the appropriate order and the base layer last. Thus, if there were no losses due to other causes then this would be an ideal way to place the stream into packets.

Unfortunately, there will be losses of packets due to other causes, and in particular it is packet loss that triggers the congestion control protocol in the first place to filter out packets purposely. Thus, a much better idea is to partition each layer into blocks and then use FEC codes on these blocks to add an appropriate amount of redundancy to protect it against packet loss due to congestion (but not due to packet filtering). With this in mind, it is clear that the base layer should be protected more than the first enhancement layer, and the first enhancement layer should be protected more than the second, etc. This is because the congestion control protocol filters packets according to the loss statistics, and for example the filtering down to the base layer will be triggered by heavier packet loss due to congestion control than filtering out just the top level enhancement layer. Thus, when the congestion control protocol filters out all the layers except for the base layer, enough protection should be added to the base layer to protect it against the packet loss rate due to congestion control that triggered that

level of filtering.

As an example, suppose that there are three layers of video, each comprising a $1/3$ fraction of the raw stream rate R . Suppose further that a layer is down to a rate of $2/3$ of R , and that a packet filtering down to a rate of $1/3$ of R . Then, the base layer should be protected with at least 10 layers should be protected with at least 1 done for example in order to have a good chance to recover the base layer from the received packets when there is a 10 congestion (and thus the filtering discards all but the packets in the base layer).

There are a lot of details yet to be worked on this subject. For example, it would make sense to have only a fixed set of packet filtering rates allowed according to the layers of the raw stream, and to come up with a reasonable way to set the block lengths for each layer and the redundancy added to each layer according to the average and maximum packet loss statistics due to congestion that would cause the filtering down to this layer.

References

- [1] P. A. A. Albanese, J. Blömer, J. Edmonds, M. Luby, M. Sudan, "Priority Encoding Transmission", 35th FOCS, 1994, *IEEE Transactions on Information Theory* (special issue devoted to coding theory), Vol. 42, No. 6, November 1996.
- [2] J. Byers, M. Luby and M. Mitzenmacher, "Accessing Multiple Mirror Sites in Parallel: Using Tornado Codes to Speed Up Downloads," *ICSI technical report TR-98-021*, July 1998, and *Infocom '99*, March 1999.
- [3] J. Byers, M. Luby, M. Mitzenmacher, A. Rege, "A Digital Fountain Approach to Reliable Distribution of Bulk Data", *ACM SIGCOMM'98*, September 1998, Vancouver, Canada.
- [4] M. Handley, S. Floyd, "Strawman Specification for TCP Friendly (Reliable) Multicast Congestion Control (TFMCC)", Work in Progress, <http://north.east.isi.edu/mjh/rmcc.ps.gz>.
- [5] A. Legout, J. Nonnenmacher, E. Biersack, "Bandwidth Allocation Policies for Unicast and Multicast Flows", *IEEE Infocomm'99*, March 1999, New York, USA.
- [6] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, V. Stemann, "Practical Loss-Resilient Codes", 29th ACM STOC, 1997.

- [7] M.Mathis, J.Semke, J.Madhavi, T.Ott, "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm", ACM CCR, V.27, n.3, July 1999.
- [8] S. McCanne, V. Jacobson, and M. Vetterli, "Receiver-driven Layered Multicast", ACM SIGCOMM'96, August 1996, Stanford, CA.
- [9] J.Padhye, V.Firoiu, D.towsley, J.Kurose, "Modeling TCP Throughput: A Simple Model and its Empirical Validation", ACM SIGCOMM'98, Vancouver, Canada.
- [10] S.Paul, K.K.Sabnani, J.Lin, S.Bhattacharya, "Reliable Multicast Transport Protocol (RMTP)", *IEEE Journal on Selected Areas in Communications*, V.15, n.3, Apr. 1997.
- [11] L.Rizzo, L.Vicisano, "Reliable Multicast Data Distribution protocol based on software FEC techniques", IEEE HPCS'97, June 1997, Chalkidiki, Greece.
- [12] T.Speakman, D.Farinacci, S.Lin, A.Tweedly, "PGM Reliable Transport protocol", IETF draft-speakman-pgm-spec-02.txt, Aug. 1998.
- [13] W.Tan and A.Zakhor, "Resilient Compression of Video for Transmission over the Internet". Proc. 32nd Asilomar Conf. Signal, Sys. and Computers, November 1998.
- [14] W.Tan and A.Zakhor, "Real-time Internet Video Using Error Resilient Scalable Compression and TCP-friendly Transport Protocol", IEEE Trans. Multimedia, To appear, June 1999.
- [15] L.Vicisano, L.Rizzo, J.Crowcroft, "TCP-like congestion control for layered multicast data transfer", IEEE Infocom'98, March 1998, San Francisco, CA.

The Loss Path Multiplicity Problem in Multicast Congestion Control

Supratik Bhattacharyya, Don Towsley, Jim Kurose
Department of Computer Science
Univ. of Massachusetts Amherst
Amherst MA 01002 USA
{bhattach,towsley,kurose}@cs.umass.edu

Abstract—An important concern for source-based multicast congestion control algorithms is the loss path multiplicity (LPM) problem that arises because a transmitted packet can be lost on one or more of the many end-to-end paths in a multicast tree. Consequently, if a multicast source's transmission rate is regulated according to loss indications from receivers, the rate may be completely throttled as the number of loss paths increases. In this paper, we analyze a family of additive increase multiplicative decrease congestion control algorithms and show that, unless careful attention is paid to the LPM problem, the average session bandwidth of a multicast session may be reduced drastically as the size of the multicast group increases. This makes it impossible to share bandwidth in a *max-min fair* manner among unicast and multicast sessions. We show that max-min fairness can be achieved however, if every multicast session regulates its rate according to the most congested end-to-end path in its multicast tree. We present an idealized protocol for tracking the most congested path under changing network conditions, and use simulations to illustrate that tracking the most congested path is indeed a promising approach.

Keywords—multicast, congestion control, multiple loss paths, max-min fairness.

1. INTRODUCTION

THE deployment of multicast services in wide-area networks is expected to lead to a proliferation of point-to-multipoint applications in the near future. Such applications will constitute a significant portion of the overall network traffic and will compete with existing point-to-point applications for network bandwidth. Hence it is necessary to control and regulate their bandwidth consumption in order to prevent network congestion.

One possible approach towards multicast congestion control is source-based rate control, in which a multicast source regulates its transmission rate in response to loss indications (e.g., NAKs) from receivers. A number of specific source-based rate control schemes have been proposed [1–3]; these represent important first solutions in a very large solution space. However, a number of fundamental issues remain open and have to be addressed by any source-based approach towards multicast congestion control. In this paper, we identify and examine two such issues.

First, the loss indications received by a multicast source from multiple receivers reflect diverse congestion conditions in various parts of the network, and have to be appropriately combined when making a single rate control decision. A transmitted packet may be lost on one or more of the many end-to-end

paths in a multicast distribution tree. The number of such loss paths is likely to increase with an increase in the number of receivers; hence the probability that the source receives *at least* one loss indication for every transmitted packet becomes high. If the source reduces its rate in response to every loss indication that it receives, its transmission rate will be severely throttled.

The second important issue concerns fairness in bandwidth sharing among unicast and multicast sessions. Multicast connections should not be allowed to usurp a large share of bandwidth since that may starve unicast connections. On the other hand, a multicast session's rate should not be throttled to the extent that its bandwidth share is drastically reduced, since that will discourage the widespread deployment and use of multicast technology.

The central issue of interest in this paper is how a multicast source combines loss indications from multiple receivers in its multicast distribution tree for rate regulation, and how such combinations affects fairness in bandwidth sharing. We analyze a family of additive increase multiplicative decrease congestion control algorithms [4], and show that, unless careful attention is paid to the existence of multiple loss paths in a multicast tree, the average bandwidth share of a multicast session may be reduced drastically as the size of the multicast group grows. Our results also indicate that it is impossible to share bandwidth in a max-min fair manner unless the problem of multiple loss paths is addressed. Our definition of max-min fairness is based on allocating bandwidth to a multicast session according to the most congested path in its multicast tree. We show that it is possible to ensure max-min fairness according to this definition only if every multicast source regulates its rate according to the most congested path in its tree, or equivalently, according to loss indications from only the "lossiest" receiver in the multicast group. We present simulation results that show that tracking the worst receiver is indeed a promising approach.

The rest of the paper is organized as follows. Section 2 presents an discussion on the problem arising out of the existence of multiple loss paths in a multicast tree, and its effect on fair bandwidth sharing. In Section 3, we describe a family of additive increase multiplicative decrease congestion algorithms and express the average session bandwidth as a function of the observed loss probability at the source for these algorithms. Section 4 describes a method for analytically deriving the loss indication probability at a multicast source for some of these algorithms. In doing so, we uncover some differences between applications where the source retransmits lost data pack-

This work was supported by the National Science Foundation under grant NCH 9506274 and by TASC under subcontract J08899-S97114. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

cts and ones where it does not do so. In Section 5, we present a number of case studies where we compute the average session bandwidth for some of these algorithms for a number of network scenarios. The results illustrate the severe degradation in a multicast session's bandwidth share when the source responds to loss indications from all receivers in the group. Section 6 presents simulation results that indicate that it is indeed possible to eliminate the problem of multiple loss paths, and to achieve max-min fair sharing of bandwidth by regulating a source's rate according to loss indications from only the worst receiver in the multicast group. Section 7 concludes the paper.

II. THE MULTICAST LOSS PATH MULTIPLICITY PROBLEM AND FAIRNESS

The most widely used unicast congestion control approach in the Internet is end-to-end control of user traffic at the transport level. In this approach, each traffic source regulates its rate based on loss (and/or delay) feedback from its receiver. The source maintains a rate (or window, as in the case of TCP [5]) parameter that is decreased multiplicatively every time a congestion feedback (e.g., loss indication) is received, and increased additively otherwise.

Let us now consider extending this approach to a multicast source, with the source adjusting its rate in response to loss indications (LIs) from receivers in its multicast group. This gives rise to two problems. The first is the problem of spatial loss correlation - a single packet loss may affect multiple receivers; hence the source may receive more than one LI for the loss. If the source reduces its rate in response to each such LI, it will have overcompensated for the single loss. One possible way of countering this problem is to have the source reduce its rate less aggressively for each individual LI. Alternatively, assuming all LIs for the same packet loss reach the source within a certain time window, the source can react to only one of them and ignore the rest [1].

The second problem arises due to the existence of multiple end-to-end paths in a multicast tree. Suppose that a multicast source reduces its rate in response to LIs from all its receivers, but reacts to no more than one LI per transmitted packet. However, a transmitted packet may be lost independently on one or more of the multiple paths in the tree. As the number of such paths increases, the probability that the source receives at least one LI per transmitted packet also increases. We refer to this problem as the loss path multiplicity (LPM) problem. In order to gain an intuitive understanding of the problem and its effect, let us consider a multicast group with n receivers, each independently experiencing a loss probability of p . Then the probability that the source receives at least one LI per transmitted packet is given by $Q = 1 - (1 - p)^n$. As $n \rightarrow \infty$, $Q \rightarrow 1$. Therefore the multicast source regulates its rate as if it were observing a single network path with loss probability Q , and the average session bandwidth is very low.

If the LPM problem reduces the bandwidth share of multicast sessions, competing unicast sessions will receive most of the available network bandwidth, resulting in unfairness in bandwidth sharing. In order to evaluate the extent of this unfairness, we introduce the following fairness criterion. First, neither unicast sessions nor multicast sessions are to be given preferen-

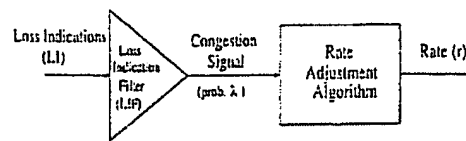


Fig. 1. General representation of FLICA algorithms

tial treatment when allocating bandwidth. Second, bandwidth is allocated to each multicast session according to the most congested path in its tree. Such a policy conforms to the widely popular notion of max-min fairness [6]. For example, suppose there is an amount of bandwidth B , available on the most bandwidth-constrained path in a multicast tree and there is one more unicast session that traverses this path. Then under the fairness definition, the multicast and the unicast session will each be allocated a share $B/2$. Consequently the multicast session would also be allocated bandwidth $B/2$ on every other path in its tree, even if there is excess capacity on those paths. The excess bandwidth on those paths is then available to other sessions that traverse those paths. Thus our policy makes no assumptions about the kind of preferential treatment to be given to any session. Also, it allocates multicast bandwidth based on the notion that a multicast session can only use as much bandwidth as is available on the most bandwidth-constrained path.

In the rest of the paper, we examine how the LPM problem may introduce unfairness according to this definition of max-min fairness. In the course of our study, we also identify a promising end-to-end approach for ensuring fairness. This approach is based on having each multicast source identify the most congested path in its distribution tree, by identifying the "lossiest" or "worst" receiver i.e., the one experiencing the highest end-to-end loss probability. The source rate is then regulated in response to LIs from only this receiver. Of course, algorithms for increasing and decreasing the source rate have to be chosen such that any two sessions experiencing the same end-to-end loss probabilities will receive equal shares of bandwidth.

III. A FAMILY OF RATE CONTROL ALGORITHMS

In this section we describe a family of additive increase multiplicative decrease algorithms (AIMD) [4], collectively referred to as FLICA (Filtered Loss Indication-based Congestion Avoidance), for regulating a multicast source's transmission rate. Each algorithm in the class decreases the transmission rate multiplicatively in response to LIs from receivers, and increases it additively in the absence of LIs. From the discussion in Section II, we observe that every LI from every receiver may not be considered for rate adjustment. The source decides which LIs to use for this purpose and filters out the rest. Let us define a congestion signal (CS) as an LI that the source uses for rate adjustment. We can identify two main components of any FLICA algorithm (Figure 1):

- a Loss Indication Filter (LIF): this determines which of the LIs received are to be considered as CSs.
- a Rate Adjustment Algorithm: an algorithm that determines how to decrease the rate when a CS is received and how to increase the rate in the absence of CSs.

The design of the LIF is policy dependent. For example, in the case of a representative-based scheme [2], where a source responds to LIs only from a subset of receivers designated as representatives, an LIF would filter out all LIs from non-representatives. Instead, the LIF may be timer-driven, letting through no more than one LI within a certain time interval. Such a time-driven LIF corresponds closely to the LTRC scheme in [1]. The LIF filter need not necessarily be located at the source, and may be centralized or distributed. For example, the representative scheme in [2] proposes that non-representative receivers suppress their LIs using backoff timers. For an active network protocol such as the one proposed in [7], filters are actually located at the active nodes inside the network and selectively forward LIs towards the source. Note that Figure 1 applies to unicast sources as well, with the LIF in that case letting through all LIs from the single receiver.

For every FLICA algorithm, the source maintains a variable r that represents the current transmission rate of the source. The value of r is adjusted in response to CSs in the following manner:

On receiving a CS, $r \leftarrow r - r/C$,
In the absence of any CS for time S , $r \leftarrow r + 1$.

where C and S are adjustable parameters. Therefore the transmission rate is reduced by $1/C$ of its current value on receiving a congestion signal (multiplicative decrease). In the absence of such signals, r is increased by 1 every S units of time (additive increase). A particular FLICA algorithm is completely defined by specifying its LIF and the values of C and S .

Let us define the congestion signal probability λ as the probability that the source receives a CS for an arbitrary transmitted packet. If B is the average session bandwidth obtained by the source under a FLICA algorithm, then the functional dependence of B on λ is given by

$$B(\lambda) = \frac{1}{\lambda S \left(0.5 + \sqrt{0.25 + \left(\frac{2}{SC-1} \right) \frac{1}{\lambda S}} \right)} \quad (1)$$

The derivation of this result is provided in [8].

IV. CONGESTION SIGNAL PROBABILITIES FOR SOME LIF POLICIES

In this section, we consider a few specific LIF policies and describe how to compute the value of λ for these policies and for a given multicast topology. Computing the value of λ is a prerequisite for analytically computing the average session bandwidth (equation (1)) attained by a session for a particular FLICA algorithm. The LIF policies that we consider here are:

- **Pass-All**: Of all the LIs received for a transmitted packet (new or retransmitted), only one is considered as a congestion signal, and this LI may be from any receiver in the multicast group.
- **Pass-K-of-N**: Given N receivers in a multicast group, K ($K \leq N$) receivers are designated as representatives. All LIs from the $N - K$ non-representatives are ignored. If one or more LI(s) are received from representatives for a transmitted packet, only one is considered as a CS.

- **Pass-Worst**: The receiver with the highest end-to-end loss probability is identified and all LIs from that receiver are considered as CSs. LIs from all other receivers are ignored. Note that Pass-All and Pass-Worst are special cases of Pass-K-of-N. However, we introduce them separately for ease of exposition.

Before we proceed with the derivation of λ for these LIFs, we need to make a distinction between two models of data delivery. The first is *reliable delivery*, where the source retransmits a data packet as many times as required, until the packet has been delivered at least once to every receiver in the multicast group. In this case, the probability of generating a LI decreases with repeated retransmissions of a packet. This must be taken into account when deriving an expression for λ . The second model of data delivery is *no-retransmissions delivery* where the source does not perform any retransmissions. Loss indications (LIs) are used in this case solely for rate adjustments. This model is applicable to continuous media applications or to reliable data transfer applications with repair servers providing repairs for lost data packets. Unlike the reliable data delivery case, the probability of generating at least one LI for a packet is now the same for every packet transmitted, since no packet is transmitted more than once by the source. We now present a method for computing the value of λ in each case.

A. Reliable Data Delivery

Let us first consider the Pass-All LIF policy. Let $\mathcal{T} = (\mathcal{M}, \mathcal{E})$ be the multicast distribution tree spanning all receivers in a multicast group, where \mathcal{M} is the set of nodes, \mathcal{E} is the set of directed edges in the tree and all receivers are attached to leaf nodes of the tree. Let $S \in \mathcal{M}$ denote the root of the tree (i.e. the node closest to the source) and let $c(n)$, $n \in \mathcal{M}$, denote the set of child nodes of node n in \mathcal{T} .

Let $R^T(n)$ denote the number of times a packet has to be transmitted to a node $n \in \mathcal{T}$, until it has been received at least once by all receivers downstream from n . Let F_n^T be the probability distribution function for $R^T(n)$ and p_n be the loss probability of a packet at node n . The expression for $F_n^T(i) = P[R^T(n) \leq i]$ is given in [9]:

$$F_n^T(i) = \begin{cases} 1 - p_n^i, & n \text{ is a leaf} \\ \sum_{u=0}^{i-1} \binom{i-1}{u} p_n^u (1 - p_n)^{i-u} \prod_{k \in c(n)} F_k^T(i-u), & \text{otherwise} \end{cases} \quad (2)$$

Therefore,

$$E[R^T(S)] = \sum_{i=0}^{\infty} (1 - F_S^T(i)) \quad (3)$$

Since this is the expected number of times that a packet will be transmitted, the expected number of times that at least one receiver will lose the packet is $E[R^T(S)] - 1$. For each of these times, the source will receive a CS. Hence the expected number of CSs generated per $E[R^T(S)]$ packets is $E[R^T(S)] - 1$, and

$$\lambda = 1 - \frac{1}{E[R^T(S)]} \quad (4)$$

Now consider the Pass-K-of-N LIF. Let $\mathcal{G} = (\mathcal{M}', \mathcal{E}')$ be the multicast distribution tree for only the representatives, where

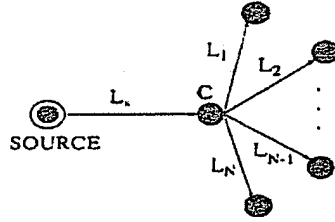


Fig. 2. Modified Star Topology. q = loss probability on L_s ; p_i = loss probability on L_i .

$\mathcal{M}' \subset \mathcal{M}$ and $\mathcal{E}' \subset \mathcal{E}$. Note that for the Pass-Worst LIF, the tree \mathcal{G} consists of the single end-to-end path from the source to the lossiest receiver. Then the expected number of times a packet has to be transmitted in order to be delivered at least once to each of the representatives is

$$E[R^{\mathcal{G}}(S)] = \sum_{i=0}^{\infty} (1 - F_S^{\mathcal{G}}(i)) \quad (5)$$

where $F_S^{\mathcal{G}}(i)$ is defined by equation (2). Hence λ is given by

$$\lambda = \frac{E[R^{\mathcal{G}}(S)] - 1}{E[R^{\mathcal{T}}(S)]} \quad (6)$$

Note that in the case that $K = N$, $\mathcal{G} = \mathcal{T}$, the Pass-K-of-N LIF reduces to Pass-All and (6) reduces to (4). We have used the above technique to compute λ for two simple topologies – a “modified star” (Figure 2) and a complete binary tree. The derivations are described in [8].

B. No-retransmission Data Delivery

For the Pass-All LIF, let us again start with an arbitrary multicast tree $\mathcal{T} = (\mathcal{M}, \mathcal{E})$ spanning all receivers in the multicast group. Let p_n be the probability of packet loss at node n , $n \in \mathcal{M}$. Let $Q_n^{\mathcal{T}}$ be the probability that a packet transmitted to node n is lost by at least one receiver downstream from n . $Q_n^{\mathcal{T}}$ is computed recursively according to the following equation:

$$Q_n^{\mathcal{T}} = \begin{cases} p_n, & n \text{ is a leaf node} \\ 1 - (1 - p_n) \prod_{m \in \text{child}(n)} (1 - Q_m^{\mathcal{T}}), & \text{otherwise} \end{cases} \quad (7)$$

Then,

$$\lambda = Q_S^{\mathcal{T}} \quad (8)$$

where S is the root of \mathcal{T} .

For a Pass-K-of-N LIF, λ is given as

$$\lambda = Q_S^{\mathcal{G}} \quad (9)$$

As in the reliable data delivery case, [8] describes how to derive λ for the modified star and the complete binary tree topologies using the above method.

V. CASE STUDIES

In this section, we study the behavior of some specific FLICA algorithms for a modified star topology under several different network loss scenarios. The metric used for evaluating the performance of the algorithms is the average session bandwidth B . B is computed using equation (1), with the value of λ computed according to the method described in Section IV. The purpose of this study is to gain insights into the effect of the LPM problem and into its possible solutions. We also study the performance of the congestion control algorithm proposed for the PGM protocol with the goal of understanding whether, and to what extent, it is affected by the LPM problem.

The FLICA algorithms studied here use different loss indication filters (LIFs) but the same rate adjustment algorithm with $C = 2.0$ and $S = 0.2$ sec.

Let the modified star (Figure 2) have $N = 50$ receivers. Let Q_i be the end-to-end loss probability for receiver i . With p_i and q as defined earlier, we then have $p_i = (Q_i - q)/(1 - q)$. Let us define the independent loss ratio as $f_i = p_i/Q_i$. This is a measure of the fraction of independent (i.e. not spatially correlated with any other receiver) loss for receiver i .

Let us first consider identical loss probabilities for all receivers, i.e. $Q_i = Q = 0.05$, $i = 1, \dots, 50$. This implies that the independent loss ratio is also the same for all receivers. Let $f_i = f$, $i = 1, \dots, 50$. Figure 3 illustrates the dependence of λ and B on f in the case of applications requiring reliable data delivery. Figure 4 shows the same for applications using no-retransmission data delivery.

We observe that for a Pass-All LIF, there is a sharp increase in the value of λ with increasing f . The effect is less significant for reliable delivery since the probability of getting a NAK decreases with repeated retransmissions of a packet. On the other hand, with a Pass-Worst LIF (in this case, any one receiver can be selected as the representative to track), there is no such sharp increase in λ . For no-retransmission delivery, λ is simply the end-to-end loss probability for any receiver; hence it remains invariant with f . Interestingly, in the reliable delivery case with a Pass-Worst LIF, λ decreases with increasing f . The reason for this is as follows. Once the tracked receiver has received a packet, it ignores all subsequent retransmissions of the same packet. Hence if any such retransmission is lost by one of the other receivers, the source does not receive a CS for it. As the spatial loss correlation decreases, there is greater chance that the tracked receiver receives a packet which one or more of the other receivers have lost. Since no CS is generated for any of the subsequent retransmissions, λ decreases.

For the Pass-All LIF, the increase in λ with f leads to a drastic reduction in the bandwidth (B) actually used by the multicast session, since B is a decreasing function of λ . Significantly, most of this reduction takes place between $f = 0.0$ and $f = 0.1$, indicating that even small amounts of uncorrelated loss can have harmful consequences for a multicast session's average bandwidth. We also observe that, with a Pass-Worst LIF, there is no such degradation in B , since λ remains more or less unchanged.

From Figure 5, we observe that B scales poorly with the number of receivers (N) for a Pass-All filter. The degradation is quite drastic even when the independent loss ratio, f , is as small as 0.1. This clearly shows the scalability problem introduced

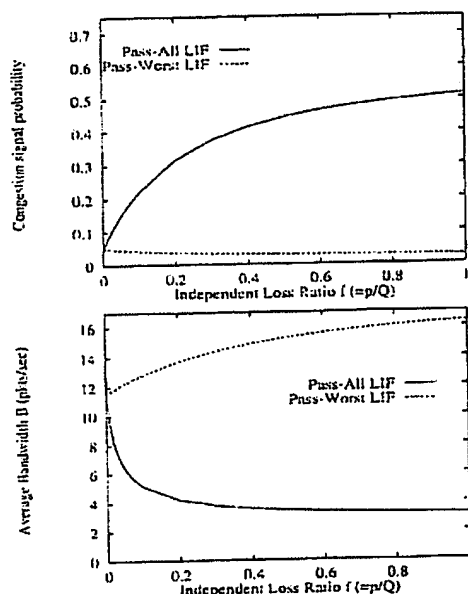


Fig. 3. Congestion signal probability (λ) and average session bandwidth (B) vs. independent loss ratio (f) for reliable data delivery with a modified star topology having $N = 50$, $Q_i = Q = 0.05$, $i = 1, \dots, 50$. Algorithm: FLICA with $C = 2$, $S = 0.2$ sec.

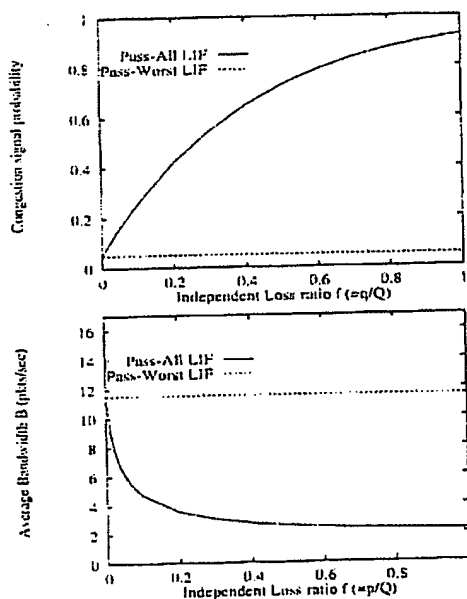


Fig. 4. Congestion signal probability (λ) and average session bandwidth (B) vs. independent loss ratio (f) for no-retransmission data delivery with a modified star topology having $N = 50$, and $Q_i = Q = 0.05$, $i = 1, \dots, 50$. Algorithm: FLICA with $C = 2$, $S = 0.2$ sec.

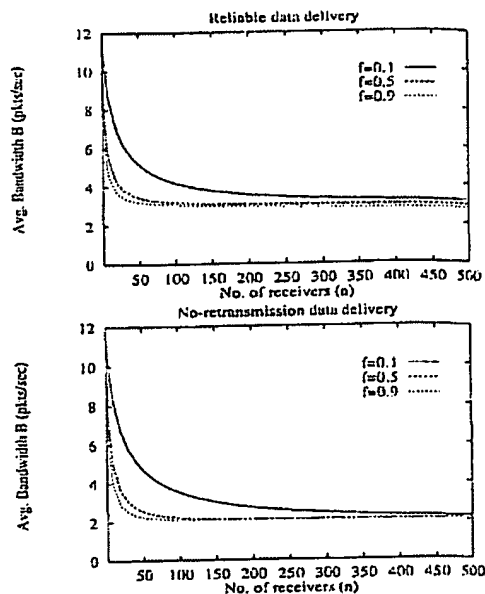


Fig. 5. Average session bandwidth (B) vs. no. of receivers (N) for (A) reliable data delivery and (B) no-retransmission data delivery with a modified star topology. $Q_i = Q = 0.05$, $i = 1, \dots, N$. Algorithm: FLICA with $C = 2$, $S = 0.2$ sec, Pass-All LIF

by loss path multiplicity for a FLICA algorithm that responds to LIs from all receivers.

We have considered a scenario where the loss probability on one arm of the modified star is higher than the others [8], and have arrived at the same conclusions as in the uniform loss case described above. However, we do not describe the details here due to space constraints.

From the results so far, we infer that the LPM problem arises from tracking LIs from a large number of receivers when not all the losses occur on the same end-to-end network path in a multicast tree. So it may be possible to use a representative scheme, where the source tracks only K of N receivers, to alleviate the LPM problem to a certain extent. We now evaluate the performance of some such schemes by considering FLICA algorithms with a Pass- K -of- N LIF for a modified star (Figure 2). Let us choose (without loss of generality) receivers $1, \dots, k$ to be the representatives. In addition to a FLICA algorithm with $C = 2$, we consider FLICA algorithms with $C = K + 1$. As the value of K increases, λ is expected to increase due to the LPM problem. For a fixed C , this has the effect of reducing B . However, if the source reacts less aggressively to each CS by using a larger value of C , then that should partially compensate for the increase in B with λ . Note that when there is a single representative, the value of C reduces to 2.

We consider three different loss scenarios. In the first case, all receivers experience the same end-to-end loss probability. We choose $Q_i = Q = 0.05$ and $f_i = f = 0.5$. Hence $q = 0.025$ and $p_i = 0.025$. We observe from Figure 6 that the degradation in B with increasing K is less severe when $C = K + 1$ than when $C = 2$, proving our intuition to be correct. In both cases

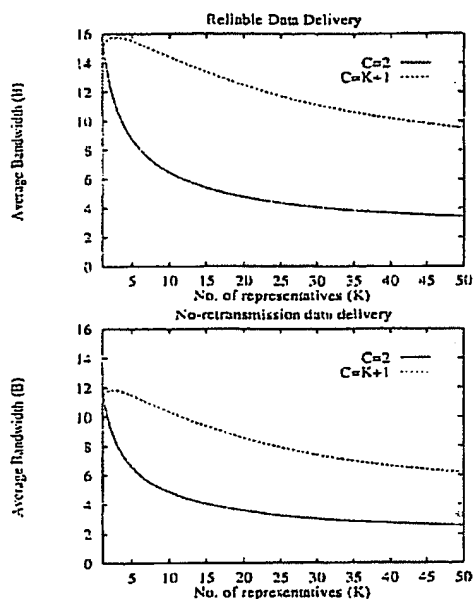


Fig. 6. Average session bandwidth (B) vs. no. of representatives (K) for (1) reliable data delivery and (2) no-retransmission data delivery with a modified star topology having $N = 50$. End-to-end loss probability for each receiver $Q = 0.05$, with the independent loss ratio $f = 0.5$.

however, as the number of representatives increases beyond a certain value, there is a considerable decrease in B . This implies that only a representative scheme with a very small number of representatives ($K \leq 5$) can counter the effect of the LPM problem.

The second loss scenario (Figure 7) differs from the first in that $p_1 = 3 \cdot p_i$, $i = 2, \dots, N$. The loss probability values chosen are $q = 0.02546$, $p_1 = 0.075$ and $p_i = 0.025$, $i = 2, \dots, N$. As expected, B decreases with K when $C = 2$. However, when $C = K + 1$, B initially increases with increasing K up to about $K = 5$, before starting to decrease. The reason is as follows. For $2 \leq K \leq 5$, the source does observe a higher λ when K increases. However this is more than compensated for by the less aggressive reaction to every individual CS, which is a result of the increase in the value of C . Hence the average session bandwidth, B , actually increases.

The third loss scenario is one in which every representative observes a significantly higher loss probability than every non-representative [8]. Due to space limitations, we omit the details here, but we arrive at similar conclusions in this case as in the two cases described above.

From the three cases studied, we conclude that the effect of loss path multiplicity can be partially alleviated by using a representative scheme. However the average session bandwidth is sensitive to the choice of representatives and the choice of the rate adjustment algorithm. These choices are difficult since they must be tailored to the observed network loss conditions. At the same time, we observe that these complications can be avoided

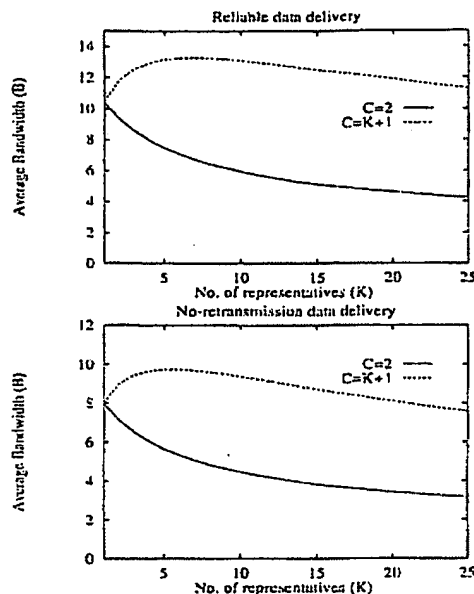


Fig. 7. Average session bandwidth (B) vs. no. of representatives (K) for (1) reliable data delivery and (2) loss-tolerant data delivery with a modified star topology having $N = 50$. Shared loss probability $q = 0.025$, $p_1 = 0.0769$, $p_i = 0.02546$, $i = 2, 3, \dots, N$.

and max-min fair sharing of bandwidth can be achieved by having each multicast source choose its worst receiver as the single representative ($K = 1$).

The LPM problem is not restricted to the family of FLICA algorithms alone. It affects any source-based rate control algorithm that reduces the source's rate in response to LIs from receivers without due consideration of the existence of multiple loss paths in a multicast tree. In order to illustrate this, we have performed case studies with the strawman congestion avoidance algorithm proposed for the PGM protocol [7]. We do not present the results here due to space constraints, but our results indicate that the impact of the LPM problem in that case is just as severe as in the case of the FLICA algorithms.

VI. A SIMULATION STUDY OF BANDWIDTH SHARING

The results in the last section indicate that the LPM problem can severely reduce the average session bandwidth of a multicast session. Representative schemes can partially alleviate the problem, but may not be able to eliminate it altogether. At the same time, tracking the worst receiver is a promising approach for ensuring max-min fair bandwidth sharing. In this section, we explore this approach more carefully through simulations.

An event-driven simulator has been used to simulate two simple networks – a two-armed star and a two-link tandem network, that are shared by a number of unicast and multicast sessions. Every session, unicast or multicast, has an infinite data source and uses a FLICA algorithm with $C = 8$ and $S = 500$ msec. We assume that data packets are never reordered, though they may be lost due to buffer overflow at the gateways. Loss indications (LIs) are in the form of negative acknowledgment (NAKs). The

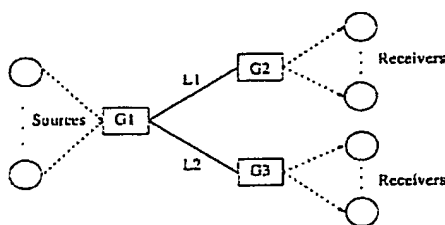


Fig. 8. Star network with two arms.

		Transmission Rate (pkts/sec)		
Simulation 1	Session Groups	Mean	Max	Min
	Multicast	29.8	30.8	29.3
	Unicast over L1	30.2	30.9	29.4
	Unicast over L2	30.3	30.9	29.4
Simulation 2	Session Groups	Mean	Max	Min
	Multicast	20.9	21.1	20.6
	Unicast over L1	20.9	21.2	20.7
	Unicast over L2	39.9	40.5	39.2
Simulation 3	Session Groups	Mean	Max	Min
	Multicast	30.0	30.2	29.4
	Unicast over L1	17.1	16.9	17.3
	Unicast over L2	30.5	30.8	30.2

TABLE I

TRANSMISSION RATES (PACKETS/SECOND) OF UNICAST AND MULTICAST SESSIONS FOR SIMULATIONS 1, 2 AND 3.

reverse path used by these NAKs is different from the forward path for data packets and NAKs are never lost or reordered. Lost data is never retransmitted by the source, hence NAKs are used only for the purpose of loss detection and rate adjustment at the source. This corresponds to the no-retransmission data delivery model described earlier. We also assume that the propagation delay on the reverse path is variable, but that the distribution of reverse path propagation delays is the same for all sessions. The bandwidth share of a session is measured in terms of the average transmission rate, r which is defined as follows. If the source transmits b packets in the interval $[t_1, t_2]$, then $r = b/(t_2 - t_1)$.

A. Star Network

The 2-armed star network consists of gateways $G1$, $G2$ and $G3$, connecting links $L1$ and $L2$, as shown in Figure 8. Each of links $L2$ and $L3$ has a bandwidth of 300 packets/second. Each gateway uses a FIFO service discipline and has a buffer size of 75. All sessions have their source connected to $G1$. Every unicast session has its receiver connected to either $G2$ or $G3$, whereas every multicast session has a receiver connected to each of $G2$ and $G3$. Thus each multicast session consists of two non-overlapping end-to-end paths over $L1$ and $L2$ respectively.

Simulation 1 involves five multicast sessions spanning $L1$ and $L2$, five unicast sessions over $L1$ and five unicast sessions over $L2$. The loss indication filter (LIF) at every multicast source is designed such that all NAKs from the receiver attached to $G2$ pass through while no NAKs from the other receiver do. In effect, every multicast session tracks NAKs from only one of two equally congested paths. All the sessions are allowed to transmit packets for 2200 seconds, with the session starting times being staggered over the first second. Table I shows the mean, maximum and the minimum value of the average transmission rates for three groups of sessions: the five multicast sessions, the five

unicast sessions over $L1$ and the five unicast sessions over $L2$. The measurement interval is taken to be (200 sec, 2000 sec). We observe that each session receives approximately the same share of bandwidth on both $L1$ and $L2$, implying that it is possible to achieve, or at least approach, max-min fair bandwidth sharing in this case.

In simulation 2, five additional unicast sessions are started on $L1$, thereby making it more congested than $L2$. Each multicast session still uses the same LIF as in simulation 1, thereby tracking the more congested path of its two paths. We observe (Table I) that on $L1$, all sessions (unicast or multicast) receive approximately an equal share (≈ 20 packets/sec) of the bottleneck bandwidth. There is less traffic on $L2$, hence more available bandwidth, however each multicast sessions is constrained to consume ≈ 20 packets/sec on *all* of its end-to-end paths. This leaves an available bandwidth of about 200 packets/sec of bandwidth available on $L2$, which is then shared equally among the five unicast sessions traversing that link. Thus by using the same control algorithm at every source and by determining a multicast session's share by its most congested path, max-min fairness has been realized.

Simulation 3 differs from simulation 2 in that the LIF for each multicast session lets through NAKs only from the receiver attached to $G3$ and filters all NAKs from the one attached to $G2$. Hence each multicast session now regulates its rate according to the less congested of its two paths. We observe that $L2$'s bandwidth is shared equally among the five multicast sessions and the five unicast sessions traversing it. But due to this, every multicast session is able to attain a rate of about 30 packets/sec over $L1$ as well. This leaves each unicast session on $L1$ with a share of about 17 packets/sec and max-min fairness is not realized. This observation emphasizes the importance of each multicast session being able to correctly identify its most congested path. However, the available bandwidth on different paths of a multicast tree may be time variant; hence, a one-time identification of the most congested path may not be not sufficient. A multicast source has to monitor all its end-to-end paths, determine which one is *currently* the most congested and then choose a receiver at the end of that path to track.

We next outline the design of an idealized protocol for doing this. In this protocol, every receiver in a multicast group to monitor packet losses on its end-to-end path and maintains a loss probability estimate p . On receiving packet i , this estimate is updated using an exponential smoothing filter:

$$p_{i+1} \leftarrow \begin{cases} (1 - \alpha)p_i + \alpha, & \text{if packet } i \text{ lost,} \\ (1 - \alpha)p_i & \text{if packet } i \text{ received.} \end{cases} \quad (10)$$

where α is the gain factor of the filter. Every receiver periodically reports the value of p to the source. The source uses a Pass-Worst LIF that remembers the identity of the receiver currently reporting the highest value of p , and allows only NAKs from that receiver to pass through. A change in the congestion condition on any end-to-end path is reflected in the value of p reported by a receiver at the end of that path. Hence the source is able to detect such changes and always regulate its rate according to the worst end-to-end path. Note that the exponential smoothing filter is one of many possible ways of estimating the loss probability; a sliding or a jumping window may be used

		Transmission Rate (pkts/sec)		
200-1200 sec.	Session Groups	Mean	Max	Min
	Multicast	21.1	21.4	20.8
	Unicast over L1	20.8	21.1	20.3
	Unicast over L2	39.7	41.2	38.8
1400-2400 sec.	Multicast	16.1	16.3	15.7
	Unicast over L1	23.1	24.2	22.0
	Unicast over L2	16.2	16.7	15.8

TABLE II
TRANSMISSION RATES (PACKETS/SECOND) OF UNICAST AND MULTICAST SESSIONS FOR SIMULATION 4.

instead. The tradeoffs in using these various approaches is an open research problem.

Simulation 4 illustrates how such an LIF can ensure max-min fair sharing of bandwidth, even under changing network conditions. In this simulation, the setting is initially identical to simulations 2 and 3, hence L1 is the more congested of the two links. However, at $t = 1200$ second, a set of ten unicast sessions are started on L2, making it more congested than L1. The value of α has been chosen as 0.01. From the result of Table II it is clear that max-min fair sharing of bandwidth both before and after the onset of additional congestion on L2. This is made possible because the LIF at each multicast source is able to always identify the most congested path. So for the interval [200 sec, 1200 sec], it identifies the receiver attached to G1 as the worst, but by $t = 1400$ sec, it has switched to the receiver attached to G2.

In addition to the star network, we performed simulations with a two-link tandem network [8] to examine the effect of spatial loss correlation on the behavior of the proposed Worst-Pass LIF. Our results indicate that even with spatial loss correlation, the bandwidth share of each multicast session is commensurate with the loss probability on its most congested end-to-end path, and max-min fairness is realized.

VII. CONCLUSIONS AND FUTURE WORK

In this paper, we have identified and studied the problem of loss path multiplicity that arises in the case of source-based multicast congestion algorithms. Our study indicates that, unless due attention is paid to the existence of multiple loss paths in a multicast tree, a multicast session's share of bandwidth may be severely reduced. As a result, max-min fair sharing of bandwidth among multicast and unicast sessions cannot be realized. Representative schemes may alleviate the LPM problem partially, but may not be able to eliminate it completely. We have also identified an approach for ensuring max-min fairness, in which every multicast source identifies the lossiest receiver (and hence, the most congested path) in its multicast tree and regulates its rate according to loss indications from that receiver. We have described an idealized protocol for identifying and tracking the worst receiver in the presence of changing congestion levels in a network.

There are many issues that remain open for future research. The design of a practical protocol for tracking the worst receiver in a multicast group requires careful consideration of the issues of estimating end-to-end loss probabilities and the timescales of congestion. A detailed discussion of these issues can be found in [8]. The issue of fairness in bandwidth sharing is a challenging

problem and criteria like TCP-friendliness ([2, 3, 10, 11]) and inter-receiver fairness [12] may have to be considered when defining fairness. The effect of feedback delay on source-based rate control techniques remains to be studied. Finally, we intend to explore how to extend our approach of tracking the worst receiver to design congestion control protocols for active networks.

ACKNOWLEDGMENTS

The first author wishes to thank Dr. Jamal Golestani of Bell Laboratories, Lucent Technologies, for his many insights on congestion control [13], and Maya Yajnik for the many fruitful discussions during the course of this work.

REFERENCES

- [1] T. Montgomery, "A Loss Tolerant Rate Controller for Reliable Multicast," Tech. Rep. NASA-IVV-97-011, West Virginia University, August 1997.
- [2] D. DeLucia and K. Obraczka, "A Multicast Congestion Control Mechanism for Reliable Multicast," Tech. Rep. 97-685, University of Southern California, Computer Science Dept., August 1997.
- [3] I. Rhee, N. Ballagura, and G.N. Kouskas, "MTCP: Scalable TCP-like Congestion Control for Reliable Multicast," Tech. Rep. TR-98-01, North Carolina State University, Department of Computer Science, January 1998.
- [4] D. Chiu and R. Jain, "Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks," *Computer Networks and ISDN Systems*, vol. 17, pp. 1-14, 1989.
- [5] V. Jacobson, "Congestion Avoidance and Control," in *Proceedings of ACM SIGCOMM*, 1988, pp. 158-173.
- [6] D. P. Bertsekas and R. G. Gallager, *Data Networks*, Prentice-Hall, Englewood Cliffs, N.J., 1992.
- [7] T. et al Speakman, "Pretty Good Multicast (PGM) Transport Protocol Specification," Internet Draft draft-speakman-pgm-spec-00.txt, January 1998, Expires on July 8, 1998.
- [8] S. Bhattacharyya, D. Towsley, and J. Kurose, "The Loss Path Multiplicity Problem in Multicast Congestion Control," Tech. Rep. 98-76, University of Massachusetts, Amherst, Department of Computer Science, July 1998.
- [9] P. Bhagawat, P.P. Mishra, and S.K. Tripathi, "Effect of Topology on Performance of Reliable Multicast Communication," in *Proceedings of IEEE Infocom*, 1994, pp. 602-609.
- [10] S. Floyd and K. Fall, "Promoting the Use of End-to-End Congestion Control in the Internet," *Submitted to IEEE/ACM Transactions on Networking*, 1998.
- [11] L. Vicisano, L. Rizzo, and J. Crowcroft, "TCP-like Congestion Control for Layered Multicast Data Transfer," in *Proceedings of IEEE Infocom*, 1998.
- [12] T. Jiang, M.H. Ammar, and E.W. Zegura, "Inter-Receiver Fairness: A Novel Performance Measure for Multicast ABR Sessions," in *Proceedings of ACM SIGMETRICS*, 1998, pp. 202-211.
- [13] S. J. Golestani, "Private Communications," 1997.
- [14] S.J. Golestani and S. Bhattacharyya, "A Class of End-to-End Congestion Control Algorithms for the Internet," in *Proceedings of ICNP*, 1998.
- [15] H. Tzeng and K. Siu, "On Max-Min Fair Congestion Control for Multicast ABR Service in ATM," *IEEE JSAC*, vol. 15, no. 3, pp. 545-556, April 1997.
- [16] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP Throughput: A Simple Model and its Empirical Verification," in *Proceedings of ACM SIGCOMM*, 1998.
- [17] B. Levine, S. Paul, and J.J. Garcia-Luna-Aceves, "Organizing Multicast Receivers Deterministically by Packet Loss Correlation," in *Proceedings of ACM Multimedia*, 1998.
- [18] M. Handley, "A Congestion Control Architecture for Bulk Data Transfer," Presentation at IRTF RMRC meeting, Cannes, France, September 1997, Expires on July 8, 1998.

A Model Based TCP-Friendly Rate Control Protocol

Jitendra Padhye[†],

Jim Kurose[†],

Don Towsley[†],

Rajeev Koodli[‡]

[†]Dept. of Computer Science,
University of Massachusetts
Amherst, MA 01003

{jitu,kurose,towsley}@cs.umass.edu

[‡]Nokia Research Center,
3, Burlington Woods Drive,
Burlington, MA 01803

rajeev.koodli@research.nokia.com

Abstract— As networked multimedia applications become widespread, it becomes increasingly important to ensure that these applications can co-exist with current TCP-based applications. The TCP protocol is designed to reduce its sending rate when congestion is detected. Networked multimedia applications should exhibit similar behavior, if they wish to co-exist with TCP-based applications [9]. Using TCP for multimedia applications is not practical, since the protocol combines error control and congestion control, an appropriate combination for non-real time reliable data transfer, but inappropriate for loss-tolerant real time applications. In this paper we present a protocol that operates by measuring loss rates and round trip times and then uses them to set the transmission rate to that which TCP would achieve under similar conditions. The analysis in [13] is used to determine this “TCP-friendly” rate. This protocol represents a *first step* towards developing a comprehensive protocol for congestion control for time-sensitive multimedia data streams. We evaluate the protocol under various traffic conditions, using simulations and implementation. The simulations are used to study the behavior of the protocol under controlled conditions. The implementation and experimentation involve over 300 experiments over the Internet, using several machines in the US and UK. Our experimental and simulation results show that the protocol is fair to TCP and to other sessions running TFRCP, and that the formula-based approach to achieving TCP-friendliness is indeed practical.

I. INTRODUCTION

Networked multimedia applications usually employ non-TCP protocols (usually UDP with some application level control) to transmit continuous media (CM)

data such as audio and video. As these applications become widespread, it becomes increasingly important to ensure that they are able to co-exist with each other and with current TCP-based applications. A key requirement of such a co-existence is the implementation of some form of congestion control that results in a reduction of transmission rate in the face of network congestion. Many current CM applications simply transmit data at the rate at which it was encoded, regardless of the congestion state of the network.

Two major considerations come into play when designing a congestion control protocol for CM applications. First, because these applications are both loss-tolerant and time-sensitive, the transmission rate might best be adapted in a manner that is cognizant of the loss resilience and timing constraints of the application [14, 18]. Second, since the applications must co-exist with TCP-based applications, the congestion control algorithms should adapt their rate in a way that “fairly” shares congested bandwidth with TCP applications. One definition of “fair” is that of TCP “friendliness” [9] – if a non-TCP connection shares a bottleneck link with TCP connections, traveling over the same network path, then the non-TCP connection should receive the same share of bandwidth (i.e., achieve the same throughput) as a TCP connection.

To develop a comprehensive CM congestion control protocol, one can begin by designing a congestion control protocol that sets the transmission rate in a TCP-friendly manner. Once such a “strawman” or baseline protocol is designed, it can then be modified to support the timeliness requirements of CM data, perhaps with some loss of “friendliness.” The design of the “strawman” TCP-friendly protocol must be flexible enough to allow such modifications. This requirement for flexibility rules out the use of TCP itself as the baseline protocol. The congestion control mech-

This material is based upon work supported by the National Science Foundation under Grant Nos. CDA-9502639 and NCR-9508274. Part of the work was done when the first author worked for Nokia Research.

anisms of TCP are tightly coupled with the mechanisms that provide reliable delivery, an appropriate combination for non-real time reliable data transfer, but inappropriate for loss-tolerant time-sensitive CM applications. In this paper, we propose a simple baseline TCP-friendly rate control protocol (TFRC) that does not couple error-recovery and congestion control, and retains sufficient flexibility for later modifications.

We present a congestion control algorithm that controls the sending rate in a manner that is roughly equivalent to that of TCP. Specifically, if a TCP connection achieves throughput X under given network conditions and measured over a given interval length, then the proposed protocol should also have a throughput of X over an interval of the same length and under the same network conditions. Note that the throughput X has to be measured over some time interval, and based on the definition of "TCP-Friendliness" proposed in [9], we assume that this interval is significantly larger than the round trip time. The actual transmission rate, X , is determined by using a model-based characterization of TCP throughput in terms of network conditions such as mean round trip time and loss rate. We base our protocol on the model proposed in [13]. In [23] the authors have proposed a similar approach for multicast congestion control, using the formula proposed in [9]. Our protocol differs from theirs in that we use a more accurate characterization of TCP and unlike [23] we do not require the use of data layering. Other TCP-friendly baseline protocols that try to mimic the major features of TCP congestion control algorithm without providing reliable delivery have been proposed [7, 17, 20, 24]. Some ongoing work, based partially on our ideas, with a focus on formula-based multicast congestion control, is also reported in [5, 6]. We discuss some of these protocols and their limitations in the next section.

We believe there are several advantages to taking a formula-based approach towards developing a TCP-friendly congestion control scheme. First, a formula based approach is *flexible*. By changing the formula, one can easily adjust the performance of the protocol. This feature can later be exploited for making the protocol sensitive to the timeliness requirement of the media being transported. In addition, if TCP and non-TCP flows are treated separately in the network (perhaps using a scheme such as [2]), then the formula-based approach can be modified to allow non-TCP flows to compete only against one another. Finally, in [23], it has been shown that such an approach is more suitable for multicasting. Thus, a formula-

based approach based on an abstract TCP characterization can be viewed as a *first step* towards developing a comprehensive solution to the problem of congestion control for CM flows.

We evaluate the protocol under various traffic conditions, using simulations and implementation. The simulations are used to study the behavior of the protocol under controlled conditions. The implementation and experimentation involve over 300 experiments over the Internet, using several machines in the US and UK. Our experimental and simulation results show that the protocol is fair to TCP and to other sessions running TFRC, and that the formula-based approach to achieving TCP-friendliness is indeed practical.

The rest of this paper is organized as follows. In Section II, we present an overview of related work reported in the literature, followed by a description of our protocol and its advantages. In Section III, we present simulation studies of our protocol. In Section IV, we present results from a "real-world" implementation of the protocol. In Section V we discuss some of our design choices. Section VI concludes the paper.

II. RATE ADJUSTMENT PROTOCOLS

Several TCP-friendly rate adjustment protocols have recently been reported in the literature [7, 17, 20, 23, 24]. Of these, [23, 24] are specific to multicast applications, while [7, 17, 20] are unicast oriented. We now briefly review each of these five schemes, describe the new TFRC protocol, and show how it overcomes some of the limitations of earlier work.

A. Previous Work

In [7], authors describe a protocol that may be classified as a "TCP-Exact" approach. They propose a protocol which manages its window size in exactly the same way as TCP does, but instead of retransmitting lost packets, it allows the user to send new data in each packet. The principle concern with this protocol is its inflexibility. Since the protocol strictly adheres to TCP window dynamics, it would be hard to modify it to take into account timeliness requirements of CM data delivery.

The TCP-friendly protocols reported in [17, 20, 23, 24] are based (either explicitly or implicitly) on the TCP characterization first reported in [9] and later formalized in [10, 12]. This characterization states that in absence of timeouts, the steady state through-

put of a long-lived TCP connection is given by:

$$\text{Throughput} = \frac{C}{R * \sqrt{p}} \quad (1)$$

where C is a constant that is usually set to either 1.22 or 1.31, depending on whether or not receiver uses delayed acknowledgments, R is the round trip time experienced by the connection, and p is the expected number of window reduction events per packet sent. Note that the throughput is measured in terms of packets/unit time. Also note that p is *not* the packet loss rate, but is the frequency of loss indications per packet sent [10]. The packet loss rate provides an *upper bound* on the value of p , and may be used as an approximation. The key assumption behind the characterization in (1) is that timeouts do not occur at all. Consequently, it is reported in [10] that (1) is not accurate for loss rates higher than 5%. As the formula does not account for timeouts, it typically *overestimates* the throughput of a connection as loss rate increases. Data presented in [10, 13] shows that timeouts account for a large percentage of window reduction events in real TCP connections, and that they affect performance significantly.

In [23] the authors propose a multicast congestion control scheme in which the data is transmitted in a “layered” manner over different multicast groups. The more layers a receiver joins, the more data it receives. In [23] the receivers compute round trip times and estimate the packet loss rate p , and use (1) to compute the “TCP-friendly” rate at which they should receive the data. Based on this estimate, and the knowledge of the layering schemes, each receiver can dynamically decide to join or leave certain multicast groups to adjust the rate at which it receives the data. In [24], the authors propose a similar scheme in which the layers have data rates that are fixed multiples of a base rate, and a TCP-like effect (additive increase, multiplicative decrease) is achieved by using strict time limits on when a receiver might join or leave a group. The analysis of the algorithm yields a throughput characterization that is similar to (1). Apart from not being TCP-friendly at loss rates above 5%, both schemes rely on data layering, which is not easy to achieve for all types of CM encodings. In addition, determining round trip times in a multicast setting is a difficult task, as noted in [23].

In [20] the authors propose a scheme that is suitable mainly for unicast applications, but may be modified for multicast applications. The scheme relies on regular RTP/RTCP reports [19] sent between the sender

and the receiver to estimate the loss rate and round trip times. In addition, they propose modifications to RTP that allow the protocol to estimate the bottleneck link bandwidth using the packet-pair technique proposed in [1]. An additive increase/multiplicative decrease scheme based on these three estimates (loss rate, round trip delay, and bottleneck bandwidth) is then used to control the sending rate. The scheme has several tunable parameters whose values must be set by the user. In addition, the scheme is not “provably” TCP-friendly, although TCP-friendliness is evidenced in the few simulations reported in the paper. In [17] the authors propose an additive increase/multiplicative decrease rate control protocol that uses ACKs (in a manner similar to TCP) to estimate round trip times and detect lost packets. The rate adjustment is done every round trip time. The authors also propose to use the ratio of long-term and short-term averages of round trip times to further fine tune the sending rate on a per-packet basis.

Although the protocols reported in [20] and [17] do not explicitly use (1) to control their rates, the work in [9, 10, 12] has shown that the relationship between loss rate and the throughput of these protocols will be similar to (1). As a result, these protocols will not be “TCP-friendly” at loss rates higher than 5%. While [20] ignores this problem, in [17] the authors mention that their work is targeted towards a future scenario in which SACK TCP [3] and RED [4] switches will be widely deployed, reducing the probability of timeouts. However, in the present Internet, TCP-Reno [21] is the predominant protocol and very few RED switches have been deployed.

In the next section we propose a new protocol that achieves TCP friendliness in a more “real world” scenario that includes competing TCP-Reno connections, drop-tail switches and diverse background traffic conditions.

B. The TFRCP Protocol

The TFRCP protocol is a rate-adjustment congestion control protocol that is based on the TCP characterization proposed in [13]. Unlike [9, 10, 12], the characterization in [13] takes into account the effects of timeouts, a consideration that is particularly important when TCP-Reno (one of the most widely deployed versions of TCP) is used with drop-tail routers, which tend to produce correlated losses. If a TCP-Reno connection encounters correlated losses, it tends to experience a significant number of timeouts [3]. In [13] the authors quantify this phenomenon and its ef-

fects on throughput. The resulting analytic characterization of TCP throughput can be stated as follows:

$$\text{Throughput} \approx f(W_{max}, R, p, B) \quad (2)$$

where throughput is measured in packets per unit time, W_{max} is the receiver's declared window size, R is the round trip time experienced by the connection, p is the loss rate (or, more accurately, the frequency of loss indications per packet sent) and B is the base timeout value [21]. A complete statement of the formula is presented in the Appendix.

There are two parts to the TFRCP protocol: a sender-side protocol and a receiver-side protocol. The sender-side protocol works in *rounds* of duration M time units. We call M the *recomputation interval*. At the beginning of each round, the sender computes a TCP-friendly rate (we will shortly describe this computation in detail), and sends packets at that rate. Each packet carries a sequence number and a timestamp indicating the time the packet was sent. The receiver acknowledges each packet, by sending an ACK that carries the sequence number and timestamp of the packet it is acknowledging. Consider an ACK for a packet whose sequence number is k . In addition to the sequence number and the timestamp, the ACK also carries a bit vector of 8 bits indicating whether or not each of the previous 8 packets ($k-7 \dots k$) was received. The sender processes these ACKs to compute sending rate for the next round. Note that each packet is ACK'd eight times, providing some protection against ACK losses.

Let us now consider the sending rate computation in detail. Consider round i . Let r_i be the sending rate for this round, R be the current round trip time estimate, and B be the estimate of the base timeout value. The number of packets to be sent in this round is $n_i = r_i * M$. The n_i packets are clocked out uniformly during the round¹. As noted earlier, packets carry a sequence number and a timestamp indicating the time the packet was sent. The sender keeps a log of all packets it has sent in this round. The log contains two entries for each packet. The first entry indicates whether the packet has been (i) received and has been acknowledged by the receiver; (ii) presumed lost; (iii) of unknown status (neither ACK'd nor yet presumed lost). We call this the "received status" of the packet. The second entry consists of a value that

is equal to the time the packet was sent plus the current base timeout value. We call this the "timeout limit" for the packet.

As the sender sends packets, it also receives ACKs from the receiver. Consider an ACK carrying sequence number k that is received by the sender at time t_k . Let the timestamp carried by the ACK be s_k . The sender updates the lost/received status of packets ($k-7 \dots k$) using the bit vector available in the ACK. The sender also updates the round trip time estimate (R) and base timeout (B) using the difference $t_k - s_k$. This update is done exactly as in TCP; see [22] for the details of the computation. At the end of the i^{th} round, the sender computes r_{i+1} as follows:

Let the current time be t_i . Let j be the packet with the smallest sequence number, whose received status was "unknown" at the end of round $i-1$, l be the last packet sent and a be the highest sequence number for which we have received an ACK. Then any packet whose sequence number lies between j and l , (both included) and whose timeout limit is less than t_i , is marked as lost. Also, any packet whose sequence number lies between j and a (both included), and whose received status is "unknown" is marked as lost. Let x_i be the number of packets marked as "received" between j and a , and let y_i be the number of packets marked as "lost" between j and a . Then:

- If $y_i = 0$, then no packets were lost and:

$$r_{i+1} = 2 * r_i$$

Hence, when no packets are lost in a round, packets are sent twice as fast in the next round. We will discuss this feature more in Section V.

- Otherwise, $y_i \neq 0$. Let $p_i = \frac{y_i}{x_i + y_i}$. In this case, the rate for round $i+1$ is

$$r_{i+1} = f(W_{max}, R, p_i, B)$$

where f is defined in (2). It is here that the analytic characterization in [13] comes into play.

The starting value r_0 , can be set to any reasonable value. We have found that for sufficiently long flows, and for reasonable values of M , the value of r_0 has little impact on the performance of the protocol. For all simulations and experiments described in this paper, we set this value to 40 packets/second. The initial values of R and B are set in a manner similar to TCP [22].

TFRCP has no built-in error recovery mechanisms. When a comprehensive congestion control protocol, based on TFRCP is developed, the applications will

¹In simulation studies, it is possible to clock out packets evenly over the entire duration of the round. This is not possible in actual implementation, due to limited accuracy of timers. We discuss this further in Section IV.

be able to choose an error control strategy that is appropriate for the given media type. An important feature of any transmission control protocol is “self-limitation” [17]. This means that if the protocol starts experiencing 100% or near 100% losses, its sending rate should be reduced to almost zero. TCP achieves this via timeouts and eventual closedown of the connection. The TFRCP protocol uses the model proposed in [13], which takes into account the effect of timeouts and automatically reduces the sending rate to very small values at high loss rates.

The key question is how frequently the sender should re-compute the rate, i.e., how to determine the value of M . In the following section we use simulations to explore various strategies for choosing M , and their impact on the performance of the protocol.

III. SIMULATION RESULTS

In this section we present simulation studies of the TFRCP protocol. The simulations are used to study the behavior of the protocol under controlled conditions. In the following section we present additional studies carried out over the Internet. We have used the ns simulator [11] for our simulations. There are two main challenges for any simulation study of this nature: first, how to select appropriate network topologies and how to effectively model the background traffic and second, how to define and measure appropriate performance metrics. Several difficulties in this regard are pointed out in [16]. Thus, before we present any simulation results, we discuss our simulation topology and our performance metrics.

A. Simulation Topology

In our simulations, we use a simple topology to uncover and illuminate the important issues; our experiments with TFRCP over the Internet test its use in “real-world” scenarios. The simulated network topology assumes a single shared bottleneck link, as shown in Figure 1. The sources are arranged on one end of the link and the receivers on the other side. All links except the bottleneck link are sufficiently provisioned to ensure that any drops/delays that occur are only due to congestion at the bottleneck link. All links are drop-tail links. Many previous studies [3, 4, 17, 20] have used similar topologies.

The problem of accurately modeling background traffic is more difficult. We consider three types of background traffic: infinite-duration FTP-like connections, medium-duration FTP-like connections and self-similar UDP traffic. The infinite-duration FTP

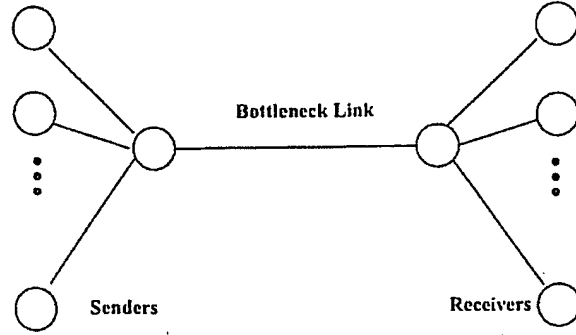


Fig. 1. Simulation Topology

connections allow us to study the steady-state behavior of our protocol. Medium-duration FTP connections introduce moderate fluctuations in the background traffic. Finally, self-similar UDP traffic is believed to be a good model for short TCP connections such as those resulting from web traffic [15, 25].

When multiple TCP connections are simulated over a single bottleneck link, the connections can become synchronized. We take two measures to prevent such synchronization. First, we start the connections at slightly different times. Second, before each packet is sent out, a small random delay is added to simulate processing overhead. These measures are applied to both TCP and TFRCP connections.

B. Performance Metrics

Recall that we view TFRCP protocol as only a *first step* towards developing a comprehensive congestion control protocol for CM data flows. Thus, we are only interested in measuring the “TCP-friendliness” of the TFRCP protocol. We define the “friendliness” metric as follows. Let k_c denote the total number of monitored TFRCP connections and k_t denote the total number of monitored TCP connections. We denote the throughput of the k_c TFRCP connections by $T_1^c, T_2^c, \dots, T_{k_c}^c$ and that of the TCP connections by $T_1^t, T_2^t, \dots, T_{k_t}^t$ respectively. Define:

$$T_C = \frac{\sum_{i=1}^{k_c} T_i^c}{k_c} \quad \text{and} \quad T_T = \frac{\sum_{i=1}^{k_t} T_i^t}{k_t}$$

The performance metric of interest is the “friendliness ratio”, F :

$$F = T_C / T_T$$

Another metric for measuring performance is the “equivalence ratio”, E :

$$E = \max(T_T / T_C, T_C / T_T)$$

Note that the value of E is always ≥ 1 . E gives a better visual representation of the closeness of the throughputs achieved by the two protocols. However, this metric will distort any trend that might be present in the ratio of the two throughputs as we vary various parameters. For example, a decreasing value of F as a function of some system parameter will not always result in a decreasing value of E . Thus, we use F as the fairness metric whenever we are interested in *trends*, and use E otherwise. It is also important that the TFRCP connections achieve fairness amongst themselves. We define the ratio:

$$FC = \frac{\max_{1 \leq i \leq k_c} T_i^f}{\min_{1 \leq i \leq k_c} T_i^f}$$

to characterize the fairness achieved among the TFRCP connections.

C. Simulation Scenarios

We now present results of performance evaluation of TFRCP protocol in various simulation scenarios.

C.1 Long duration flows with constant bottleneck bandwidth

In this scenario we consider traffic made up entirely of equal numbers of infinite-duration TCP connections and infinite-duration TFRCP connections. All connections always have data to send. All connections start at the beginning of simulation and last until the simulation ends. The aim here is to study steady state behavior of TFRCP protocol. If TFRCP performs well (i.e., in a TCP-friendly manner), the TCP and TFRCP connections should see approximately the same throughput.

We vary the total number of flows in the network between 10 and 50. Half of these connections are TCP connections and the rest are TFRCP connections. The initial sending rate, r_0 , for all TFRCP connections was set to approximately 40 packets/second. The bottleneck bandwidth is held constant at 1.5Mbps, and the bottleneck delay is set to 50ms. This roughly simulates a situation in which a number of connections share a T1 link. As the number of flows grows, the window sizes of individual TCP connections shrink, increasing the probability of timeouts. In such circumstances, the congestion control protocols proposed in [17, 20] are not be able to guarantee fairness.

We consider three different ways to determine how frequently TFRCP should recompute its rate:

- Fixed recomputation interval, i.e. we use a fixed value for M . We call this strategy S1.
- The recomputation interval is a fixed multiple of round trip time. If at the beginning of round i the round trip time is rtt_i , then the next recomputation is performed after $K * rtt_i$ time units, where K is constant. We call this strategy S2.
- The recomputation interval is calculated at the beginning of each round, and is set to sum of two numbers, one of which is a constant while the other is chosen from a uniform random distribution. This strategy will further prevent TFRCP connections from synchronizing with each other. We call this strategy S3.

In Figure 2(a) we present simulation results for the case in which the TFRCP protocol uses strategy S1, with five values of M between 2 and 5 seconds. The length of each simulation was 1000 seconds, and the throughput of all connections was measured at the end of the simulation. Each data point is an average of three experiments. It can be seen that with steady state background traffic, the protocol is able to maintain a friendliness ratio close to 1.

In Figure 2(b) we present simulation results when TFRCP protocol uses strategy S2, with four values of K between 10 and 60. We notice that as the load on the network increases, the resulting TFRCP behavior is more aggressive than TCP. As the load on the network increases, the round trip time experienced by each flow also increases. As a result, each TFRCP flow re-computes its rate less frequently. TCP reduces its transmission rate multiplicatively every time it encounters a loss, and increases it only additively in case of no loss, thus the slowness of response of TFRCP flows to react to losses hurts the throughput of TCP connections. Thus TFRCP is more aggressive, and clearly S2 is not an appropriate strategy for deciding recomputation intervals.

In Figure 2(c) we present simulation results where TFRCP protocol uses strategy S3. For each line we use a different constant and a different uniform random distribution: $0.3 + [0, 5.4]$, $1.5 + [0, 3]$ and $2.7 + [0, 0.6]$. For this simulation study, all TFRCP connections were started simultaneously. It can be seen that in this third case the protocol is able to maintain a friendliness ratio close to 1.

We have performed simulations with other bottleneck delays and observed similar results. In the rest of this section we only present results using strategy S1. We do this for two reasons. First, strategy S1 is the simplest strategy. The goal of this paper is to

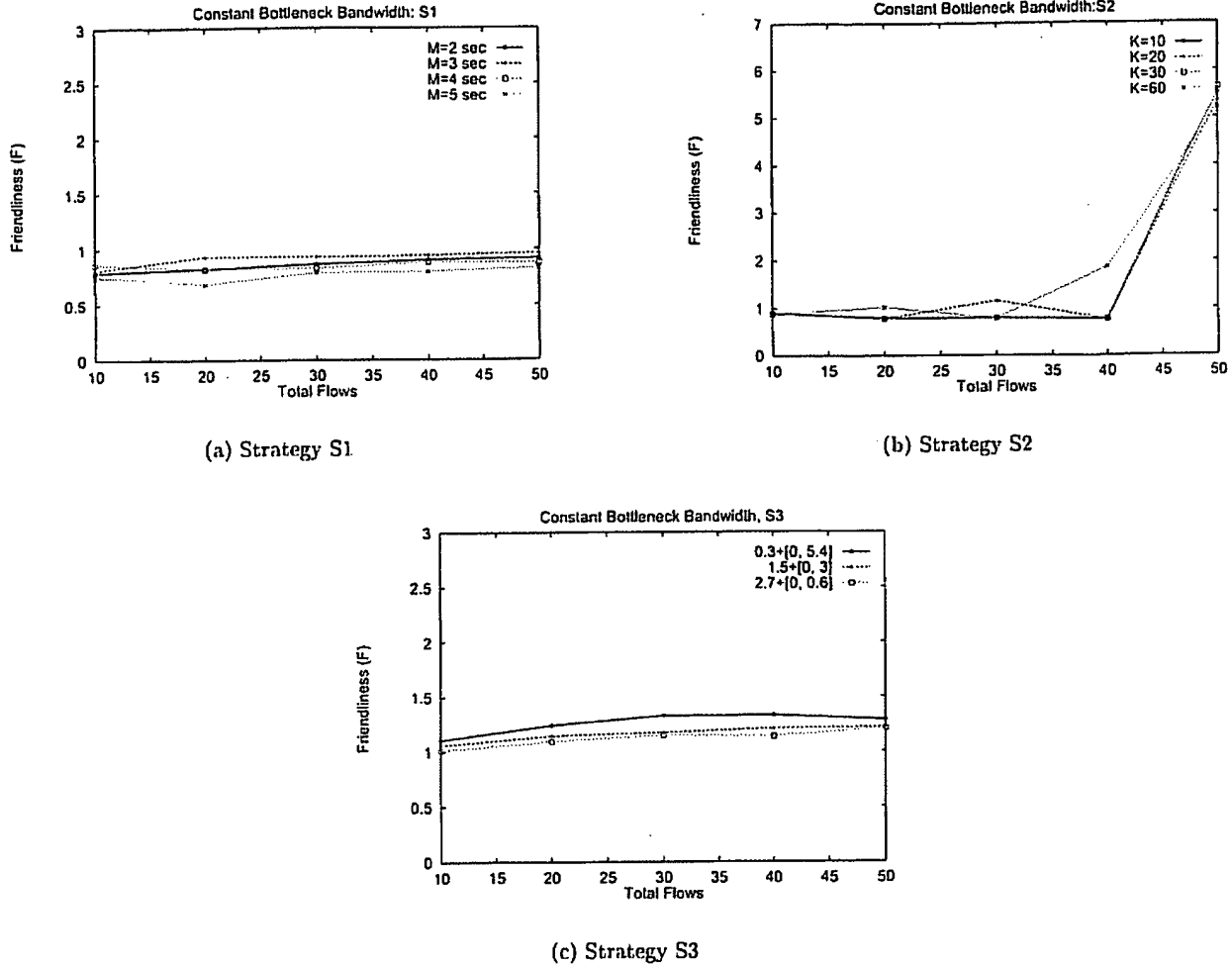


Fig. 2. Constant Bottleneck bandwidth, Bottleneck Delay 50ms

present TFRCP protocol as a baseline policy; use of a simple policy to decide the recomputation interval is consistent with that goal. Second, the question of selecting the appropriate recomputation interval requires more complex answers than the three simple strategies described here. The recomputation interval must be short enough to allow TFRCP to be responsive, while at the same time it must be large enough to allow the loss rate measurements to be meaningful. This question is currently under research [5, 6]. Thus, it is appropriate to restrict the baseline protocol described here to the simplest strategy.

Recall that the TFRCP connections should be fair to each other as well. In Figure 3 we plot the value of FC when the TFRCP protocol uses strategy S1. It can be seen that the TFRCP protocol achieves acceptable fairness among TFRCP connections in most

cases.

C.2 Long duration flows with constant bottleneck bandwidth share

In this scenario, the traffic is made up of infinite-duration TCP connections and infinite-duration TFRCP connections. All connections start at the beginning of the simulation and last until the end. We vary the total number of flows in the network between 10 and 50. The bottleneck bandwidth is computed by multiplying the total number of flows by 4Kbps. The buffer size at the bottleneck link was set in each case to four times the bandwidth-delay product. These settings of packet and buffer sizes allow the TCP connections to have "reasonable" window sizes [17] and exhibit the full range of behavior such as slow start and congestion avoidance. Each experiment is repeated

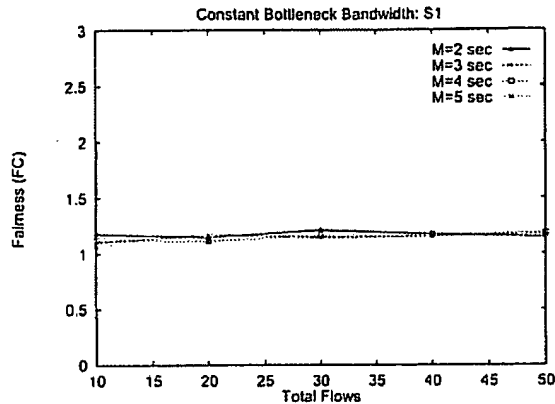


Fig. 3. Friendliness among TFRCP connections

for various values of the re-computation interval, M . The initial sending rate, r_0 , for all TFRCP connections was set to approximately 40 packets/second. In Figures 4(a) and 4(b) we plot F and FC (TCP-friendliness and Fairness among TFRCP connections) for this scenario when the bottleneck delay was 50ms and the TFRCP connections used strategy S1. It can be seen that with steady state background traffic, the protocol is able to maintain a friendliness ratio close to 1, and the TFRCP connections are fair among themselves as well. We performed simulations with bottleneck delay of 20ms and 100ms as well (not shown here), and found that for small bottleneck delays, TFRCP behaves more aggressively than TCP. We conjecture that this is due to the fact that with lower round trip times, TCP reacts to losses and small changes in traffic fluctuations more quickly. At higher round trip delays (100ms) the performance of the TFRCP protocol for small values of M (< 3 seconds) shows high variance, as the protocol is unable to gather sufficient samples to estimate loss rates accurately.

C.3 Dynamically Arriving Medium-duration FTP Connections

In this simulation scenario, we study the effect of “slow” changes in the background traffic. Recall that in the simulations described so far, traffic consisted of infinite TCP and TFRCP connections. We now consider the case that there is one infinite-duration TCP connection, one infinite-duration TFRCP connection, and additional traffic consisting of dynamically arriving TCP connections, each of which transfers a fixed amount of data. In computing F , we consider only the two infinite-duration connections. The

bottleneck link bandwidth is set to 1.5Mbps and the bottleneck delay is set to 50ms. The duration of simulation is 1000 seconds. The amount of data transferred by each background connection is chosen from a uniform distribution. The interarrival times for the medium-duration FTP connections are chosen such that on average a constant number of background connections will be active. A higher average number of background connections leads to more fluctuations in the background traffic, and in addition, the window size of each TCP connection tends to be smaller (due to a smaller bandwidth share), increasing the possibility of timeouts. We are interested in the performance of TFRCP protocol as the average number of background connections change. For graphs in Figures 5(a) and 5(b) the data transferred by each connection is chosen from $[0, 80KB]$ (average 40KB) and $[0, 160KB]$ (average 80KB), respectively.

The results in Figure 5 show that TFRCP maintains a friendliness ratio of approximately one with a re-computation interval $M = 2$ seconds. The ratio decreases as the re-computation interval becomes larger. We conjecture that this behavior is due to the nature of the background traffic. As old connections terminate and new ones start, there are small periods of time during which the background traffic decreases slightly as the new connections go through their slow start phase. TCP is better able to take advantage of these small drops in the background traffic, due to its faster feedback mechanism. The TFRCP connection changes its sending rate only every M seconds, and hence is unable to take advantage of short-term drops in the background traffic.

C.4 ON/OFF UDP traffic

In this simulation scenario, we model the effects of competing web-like traffic (very small TCP connections, some UDP flows). It has been reported in [15] that WWW-related traffic tends to be self-similar in nature. In [25], it is shown that self-similar traffic may be created by using several ON/OFF UDP sources whose ON/OFF times are drawn from heavy-tailed distributions such as the Pareto distribution. Figure 6 presents results from simulations in which the “shape” parameter of the Pareto distribution is set to 1.2. The mean ON time is 1 second and the mean OFF time is 2 seconds. During ON times the sources transmit with a rate of 12Kbps. The number of simultaneous connections is varied between 20 and 80. The simulation was run for 25000 seconds. As in the previous subsection, there are two monitored con-

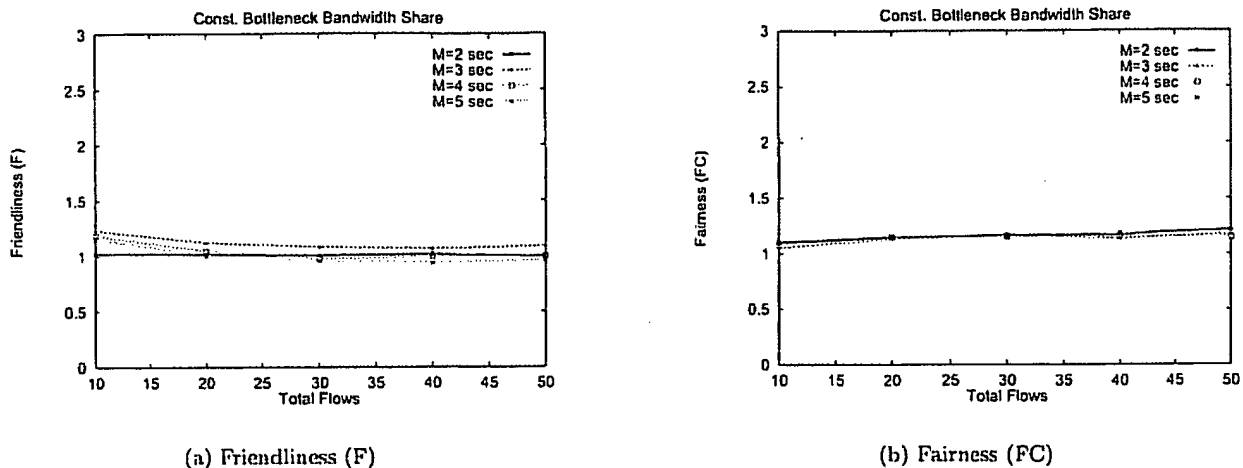


Fig. 4. Constant Bottleneck bandwidth share, Bottleneck delay 50ms

nections, an infinite TCP connection and an infinite TFRCP connection (i.e. $k = 2$). The bottleneck link bandwidth is set to 1.5Mbps and the bottleneck delay is set to 50ms. From the results in Figure 6, we can see that the TFRCP protocol is still relatively fair. The fairness index again decreases as the re-computation interval, M , increases. We believe that this is due to the fact that the TFRCP connection recomputes its rate only after every M time units. Hence, it can not increase its sending rate during the small periods of time in which the background traffic drops in intensity. Results for other values of the shape parameter are similar.

D. Summary of simulation results

The simulations results presented in this section show that the TFRCP protocol is "TCP-friendly" under a wide variety of traffic conditions. We found that the strategy to use a fixed value for re-computation interval (M), works well for a wide variety of traffic conditions. While the simulation study is based on several different traffic scenarios, it is important to observe the performance of the protocol in real world. In the next section we discuss the implementation and present results based on experiments carried out over the Internet.

IV. IMPLEMENTATION AND EXPERIMENTAL RESULTS

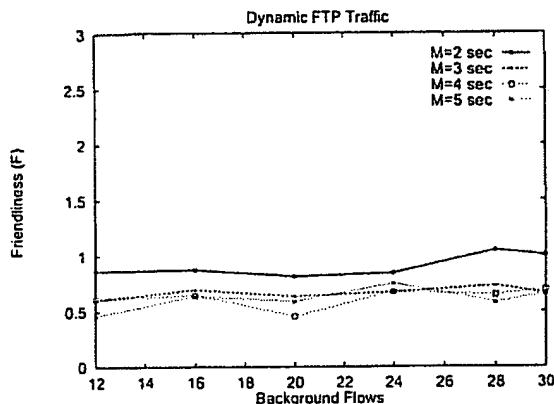
As noted in [16], simulating an Internet-like environment is very difficult. It is thus essential to test protocols like TFRCP via implementation and exper-

imentation in a real-world setting. Our goal here is thus to show that the approach is practical, and that performance of the protocol under real-world conditions is comparable to that observed in the simulations. We have implemented a prototype version of our protocol and tested it on several Unix systems. In this section, we first describe the implementation, and discuss some of the difficulties encountered. We then present the results from over 300 experiments performed using this implementation.

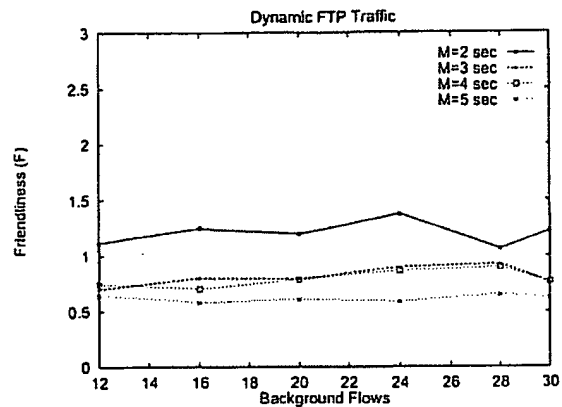
A. Implementation

Our implementation of TFRCP is done in user space, on top of UDP. The sender side of TFRCP runs as two processes, one sending the data and the other receiving ACKs. The two processes communicate via shared memory. An earlier attempt to implement the protocol using multiple threads failed, as the *p-threads* package could not provide sufficiently accurate scheduling control to avoid starving either the sender or the receiver thread. We were able to reuse much of the *ns* simulation code for the actual implementation. However, we encountered three important problems during the implementation:

- A significant problem in any actual implementation is the accuracy of the various timers involved. For simulation purposes, we could time out packets with arbitrary precision. This is not possible in an actual implementation, as the timers are neither arbitrarily accurate nor are they free of overheads. In some of our early experiments we found that when using the `gettimeofday` and `select` system calls, we could



(a) Average transfer 40KB



(b) Average transfer 80KB

Fig. 5. Friendliness (F), Dynamically arriving FTP connections

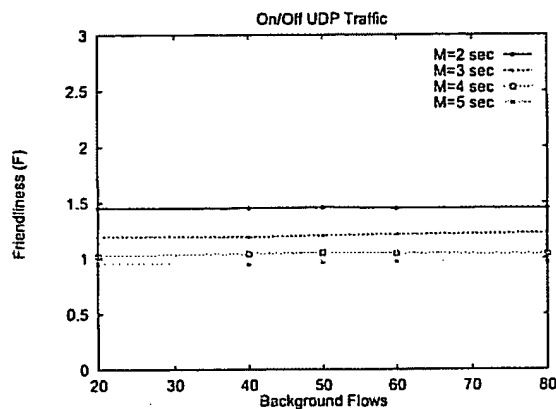


Fig. 6. Friendliness(F), ON-OFF UDP Traffic, Shape = 1.2

not control inter-packet interval more accurately than within several milliseconds. While we could achieve better accuracy using busy waiting in the process that sent the packets out, this can possibly starve the process that receives ACKs. On a FreeBSD machine used in this study, busy waiting caused other problems that forced us to use `gettimeofday` and `select` system call in our FreeBSD implementation. Due to these difficulties, it is not possible to clock packets out smoothly over the duration of each *round*, as mentioned in Section II-B. Instead, we send packets out in small bursts. Consider round i . Let R be the round trip time estimate at the beginning of this round, and r_i be the sending rate. The duration of the round is M time units. Then, the round is divided into bursts of duration R each. The number of bursts is thus, $b_i = M/R$. In each burst, n_i/b_i (rounded to nearest

integer) packets are sent back-to-back, followed by a silence period of R time units.

- Another important problem was the accuracy of the round trip times. The TFRCP protocol begins measuring the round trip time for each packet as soon as it is handed to the kernel socket using `sendto`. Thus, our round trip times include the time each packet spent waiting in the kernel buffers (similarly for ACKs). Thus, our estimate of round trip times is higher than that of the in-kernel TCP's. In addition, due to additional difficulties with timers, we had to restrict the protocol to transmit *at least* one packet per round trip time.

- In our simulation studies, it was easy to ensure that the packet sizes for TCP and TFRCP connections were the same. It is more complicated to ensure this in practice. Our implementation currently does not

Hostname	Domain	Operating System
alps	cc.gatech.edu	SunOS 4.1.3
bmt	cs.columbia.edu	FreeBSD 2.2.7
edgar	cs.washington.edu	OSF1 3.2
manic	cs.umass.edu	IRIX 6.2
maria	wustl.edu	SunOS 4.1.3
newton	nokia-boston.com	SunOS 5.5.1
sonic	cs.ucl.ac.uk	SunOS 5.5.1
void	cs.umass.edu	Linux 2.0.30

TABLE I
HOSTS USED FOR EMPIRICAL STUDIES

employ any path MTU discovery algorithm, nor does it change the size of outgoing packets. For each experiment described in the next section, the packet size is held constant, determined by the MTU discovered by TCP in previous experiments between the same two hosts. While we have found that we seldom had problems with the MTU value, it is hard to quantify the effects of constant packet size on throughput.

As a result of the implementation considerations noted above, we expect the results from implementation experiments to differ somewhat from the simulation experiments. However, we can still use the implementation to corroborate the intuition gained through our simulations, to examine TFRCP performance in real-world setting, and to provide a starting point for a more refined implementation.

B. Experimental Results

The hostnames, domains and operating systems of the machines used for the implementation study are listed in Table I. To measure the fairness of TFRCP compared to TCP, we performed the following experiment. We established two connections between a pair of hosts. One of these connections was controlled by the TFRCP protocol, while the other was controlled by the TCP protocol. Both connections ran simultaneously, and transferred data for 1000 seconds, as fast as possible. The length of the recomputation interval, M , for the TFRCP connection was set to 3 seconds; the receiver's declared window size, W_{max} , was set to 100 packets; and the initial sending rate, r_0 , was set to approximately 40 packets/second. The throughput of the two connections was measured in terms of number of packets transferred in these 1000 seconds. Let us denote these throughputs T_C and T_T respectively.

Figures 7(a)-7(c) show the results when the senders

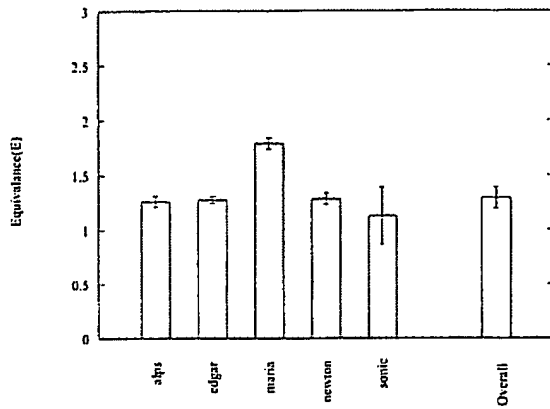
were *void*, *manic* and *bmt* respectively. Since we are not interested in trends along the x -axis, we use E as our performance metric. For each of the first five bars, the x -axis shows the receiver. To plot this graph, at least 15 experiments were performed between the sender and the receiver at random times during the day and night², and for each experiment the value of E was computed. It is suggested in [8] that data from such experiments should be represented by its median, and that the variation be represented by the semi-inter quartile range (SIQR), defined as half of the difference between the 25th and the 75th percentiles of the data set. Thus, the height of each bar is the median of that data set, while the bar represents the SIQR, centered about the median. The last bar represents the median and the SIQR of all experiments.

It can be seen that in most cases, the TFRCP protocol achieves a throughput that is within 35-50% of the TCP throughput and that the difference seldom exceeds 75%. The median of all three data sets taken together is 1.448 and the SIQR is 0.275. There are many possible reasons for the observed difference between the TCP and TFRCP throughputs. Some variation is unavoidable – we have found that the throughput of two simultaneous TCP connections between the same hosts can differ by as much as 10%. Additional variation results from the various implementation difficulties described earlier. And finally, one must remember that the formula described in [13] is only an approximation.

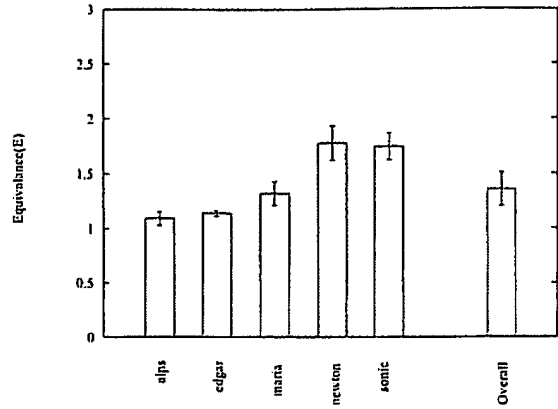
Figures 7(a)-7(c) are based on throughputs that have been computed over the entire duration of the experiment (i.e., 1000 seconds). It is also interesting to compare the difference in TCP and TFRCP as a function of time, and over shorter intervals of time. Such a comparison illustrates how well the TFRCP protocol performs at various time scales. In Figure 8, we plot the throughput of the TFRCP and the TCP connections between *manic* and *edgar*, measured every 6, 12, 24 and 48 seconds respectively. It can be seen that TFRCP tracks variations in throughput of the TCP connection quite well, at various time scales.

To measure the sensitivity of the protocol to the interval over which we measure the loss rate and update the sending rate (i.e. the value of M), we performed several data transfers between the same sender-receiver pair, using different measurement intervals. We now use F as our fairness metric, as we are interested in the trend in the performance metric

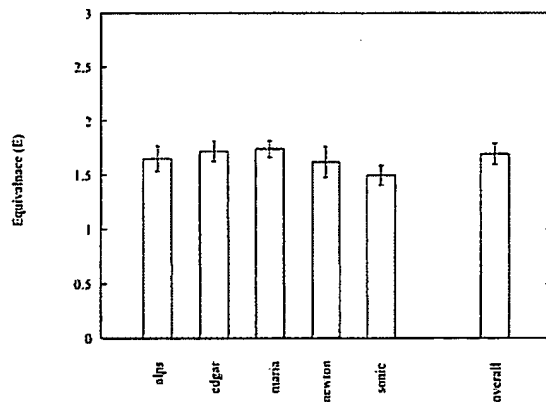
²Experiments with *bmt* as a sender were performed only during the day.



(a) Sender: void



(b) Sender: manic



(c) Sender: bmt

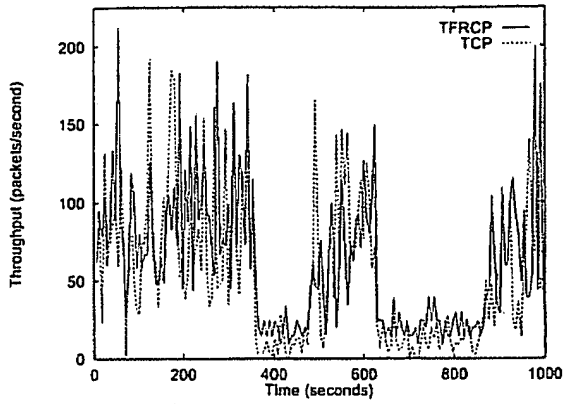
Fig. 7. Experimental Results

as we vary M . In Figure 9 we show the results of one such study, performed between *void* and *alps*. The measurement interval was varied between 2 and 10 seconds. For each value of measurement interval, at least 20 experiments were conducted at random times. The data points are the medians of the throughput ratios, and the error bars represent the SIQR. One can see that as the measurement interval grows larger, the TFRCP protocol becomes less aggressive. This result is consistent with the simulation results presented in the previous section. From the results presented in this section, we can conclude that the protocol indeed performs well in a real world setting, despite the limitations and difficulties imposed by various implementation issues.

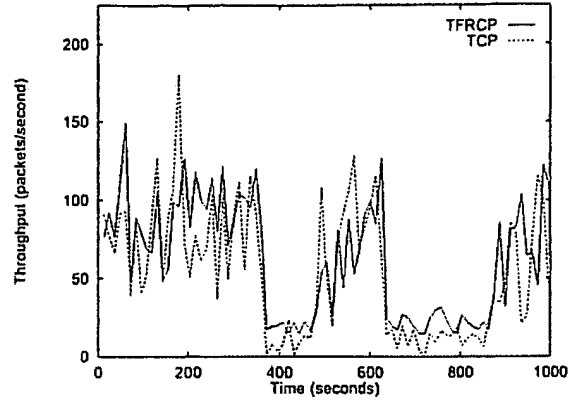
V. DISCUSSION OF PROTOCOL FEATURES

In this section we discuss the impact of some of the design choices made while simulating and implementing TFRCP.

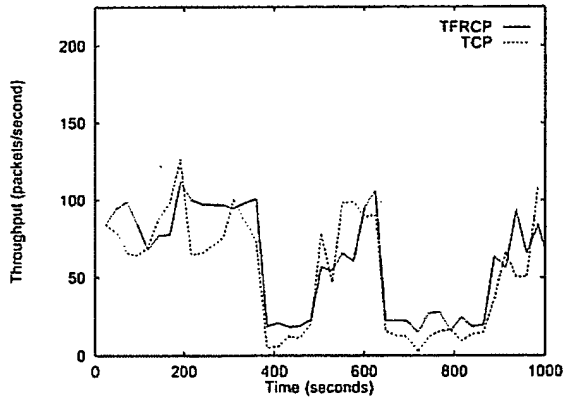
Recall that TFRCP doubles its sending rate when no packets are lost in an entire recomputation period, since the formula in (2) is not valid for zero loss rate. During periods of no loss, the TCP window grows linearly (ignoring the initial slow start period), by one every RTT. Since the sending rate is proportional to the window size, one can say that the sending rate of TCP grows linearly during periods of no loss. We found that when we try to mimic this linear increase behavior in TFRCP, the protocol performed poorly (i.e., the friendliness ratio was higher). This is due to the fact that in most of our simulations and Internet



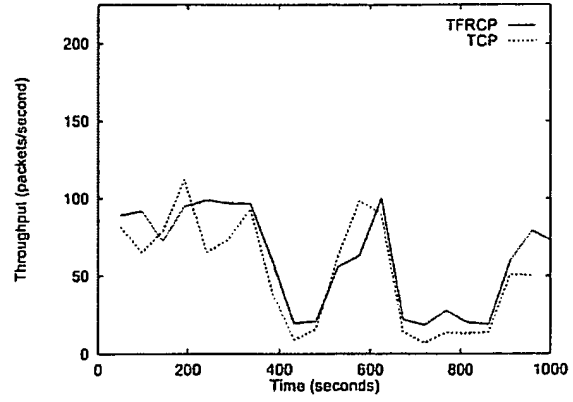
(a) Throughput every 6 seconds



(b) Throughput every 12 seconds



(c) Throughput every 24 seconds



(d) Throughput every 48 seconds

Fig. 8. Throughput at various timescales

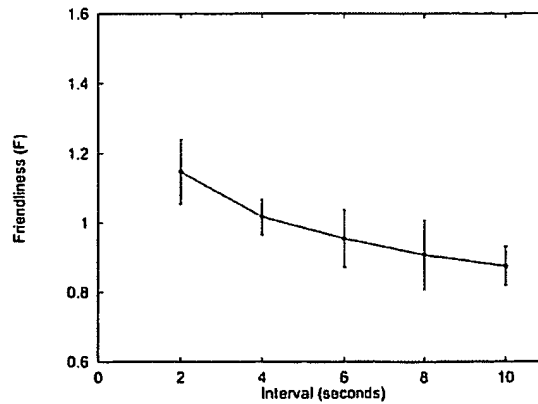


Fig. 9. Sensitivity to measurement interval. void-alps

experiments, the fair share of the TFRCP connection tended to be relatively small. If, after a recomputa-

tion interval of no loss, we increased the rate in a linear fashion, the relative change in the rate was very high

(e.g., if no loss occurred during a three-second period, and if RTT was 100ms, the rate would increase by $3/0.1 = 30$). This led to very high losses in the next round, which in turn dropped the sending rate to a very low value, leading again to a no-loss, or low-loss, period. This oscillatory behavior was detrimental to the performance of the protocol. Doubling the sending rate seems to offer a good compromise between responsiveness (ramping up the sending rate quickly) and avoiding oscillatory behavior.

The value of W_{max} can significantly affect the throughput computed using the formula in (2) at low loss rates. While in the simulations studies it is easy to ensure that competing TCP and TFRCP flows had the same value for W_{max} , this is hard to ensure in practice. We note that this problem is inevitable whenever flow control is employed: two TCP connections experiencing same network conditions, but having different values for W_{max} , will have different throughputs.

Another design issue is how to set the initial value for r_0 . For the simulation and implementation results reported in this paper, we set this value to approximately 40 packets/second. As long as the recomputation interval M was small compared to the time over which the friendliness or equivalence was being measured, the value for r_0 had little impact on the performance of TFRCP.

As mentioned in Section IV, timer inaccuracies and overheads force us to send packets out in small bursts, instead of clocking them out evenly over the duration of each *round*. The impact of this burstiness on performance of TFRCP protocol is hard to quantify. On one hand, one may imagine that burstiness would lead to slightly higher loss rates for TFRCP connection, forcing the throughput down. On the other hand, traffic from a TCP flow is somewhat bursty as well [3]. Thus the impact of bursty nature of TFRCP flow on friendliness ratio is hard to judge.

It should be noted that the formula in (2) is not valid for certain network scenarios, such as TCP connections running over modem lines with large dedicated buffers [13]. This implies that the TFRCP protocol would not work well in these situations either. We are currently working on solutions to this problem. We also note that TFRCP reacts to changes in network conditions only every M time units (i.e. the duration of recomputation interval). If the network traffic conditions change on a faster time scale, the difference between the throughput of a TCP connection and a TFRCP connection experiencing similar

network conditions may be significant. Under such dynamic conditions, obtaining accurate loss estimates and round trip times can be problematic. We note that in real-world testing, Figures 7(a)- 7(c), we have found that the protocol works well with a recomputation interval of three seconds. One may question if achieving TCP-Friendliness at large time granularities is useful at all. We would like to point out that multiple TCP connections going over the same network path need not achieve same throughput on a time scale comparable to the round trip time. Thus, fairness needs to be measured over time intervals longer than a few round trip times. One must also note that very short TCP connections such as HTTP transfers, do not achieve friendliness even among themselves. Hence, we have restricted ourselves to achieving fairness between long term TCP and TFRCP connections. We believe that as long as the duration of a flow is significantly larger than M , the TFRCP protocol achieves this goal.

VI. CONCLUSIONS AND FUTURE WORK

In this paper we have presented a TCP-friendly rate adjustment protocol. The protocol achieves TCP-friendliness by changing its sending rate, based on TCP characterization developed in [13]. using the measured loss rate and round trip times. In addition to studying the protocol through simulations, we implemented a prototype version of the protocol and tested it with experiments over the Internet. The results of both simulation and implementation experiments show that the protocol is able to achieve throughputs that are close to the the throughput of a TCP connection traveling over the same network path. Thus, we conclude that formula-based feedback-loop approach to congestion control and achieving TCP-friendliness is indeed practical.

We have identified several avenues for future work. We are currently working on developing better techniques for loss rate estimation. We plan to refine the implementation of the protocol, especially the implementation of various timers. We also plan to investigate if any other throughput formulas can be used in the feedback loop, and their impact on performance of the protocol. Above all, we are working towards developing a comprehensive protocol for congestion control of continuous media flows. The protocol will take into account the effects of limited buffer space available at the sender and the receiver, along with the timeliness requirements and loss tolerance of the specific media being sent.

REFERENCES

- [1] J. Bolot and A. Vega-Garcia. Control mechanisms for packet audio in the Internet. In *Proceedings IEEE Infocom'96*, 1996.
- [2] D. Clark and J. Wroclawski. An approach to service allocation in the internet. IETF draft-clark-diff-svc-alloc00.txt, July 1997.
- [3] K. Fall and S. Floyd. Simulation-based comparisons of Tahoe, Reno, and SACK TCP. *Computer Communication Review*, 26(3), July 1996.
- [4] S. Floyd and V. Jacobson. Random Early Detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4), August 1997.
- [5] M. Handley and S. Floyd. Strawman Specification for TCP Friendly (Reliable) Multicast Congestion Control (TFMCC). Unpublished Manuscript. <http://www.east.isi.edu/RMRG/newindex.html>.
- [6] M. Handley and S. Floyd. TCP-Friendly Simulations. Unpublished Manuscript.
- [7] S. Jacobs and A. Eleftheriadis. Providing Video Services over Networks without Quality of Service Guarantees. In *World Wide Web Consortium Workshop on Real-Time Multimedia and the Web*, 1996.
- [8] R. Jain. *The Art of Computer Systems Performance Analysis*. John Wiley and Sons, 1991.
- [9] J. Mahdavi and S. Floyd. TCP-friendly unicast rate-based flow control. Note sent to end2end-interest mailing list, Jan 1997.
- [10] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The macroscopic behavior of the TCP congestion avoidance algorithm. *Computer Communication Review*, 27(3), July 1997.
- [11] S. McCanne and S. Floyd. ns-LBL Network Simulator, 1997. Obtain via <http://www.nrg.ee.lbnl.gov/ns/>.
- [12] T. Ott, J. Kemperman, and M. Mathis. The stationary behavior of ideal TCP congestion avoidance. <ftp://ftp.bellcore.com/pub/tjo/TCPwindow.ps>.
- [13] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: A simple model and its empirical validation. In *Proceedings of SIGCOMM'98*, 1998.
- [14] C. Papadopoulos and G. Parulkar. Retransmission-based error control for continuous media applications. In *Proceedings of NOSSDAV'96*, 1996.
- [15] K. Park, G. Kim, and M. Crovella. On the relationship between file sizes, transport protocols and self-similar network traffic. In *Proceedings of ICNP'96*, 1996.
- [16] V. Paxson and S. Floyd. Why we don't know how to simulate the Internet. In *Proceedings of the 1997 Winter Simulation Conference*, 1997.
- [17] R. Rejaie, M. Handley, and D. Estrin. An end-to-end rate-based congestion control mechanism for realtime streams in the internet. In *Proceedings of INFOCOMM '99*, 1999.
- [18] I. Rhee. Error control techniques for interactive low-bit rate video transmission over the internet. In *Proceedings of SIGCOMM'98*, 1998.
- [19] H. Schulzrinne, S. Casner, R. Fredrick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 1889.
- [20] D. Sisalem and H. Schulzrinne. The Loss-Delay Based Adjustment Algorithm: A TCP-Friendly Adaption Scheme. In *Proceedings of NOSSDAV'98*, 1998.
- [21] W. Stevens. TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. RFC2001, Jan 1997.
- [22] W. Stevens. *TCP/IP Illustrated, Vol.1 The Protocols*. Addison-Wesley, 1997. 10th printing.
- [23] T. Turetti, S. Parisi, and J. Bolot. Experiments with a layered transmission scheme over the Internet. Technical report RR-3296, INRIA, France.
- [24] L. Vicisano, L. Rizzo, and J. Crowcroft. TCP-like congestion control for layered multicast data transfer. In *Proceedings of INFOCOMM'98*, 1998.
- [25] W. Willinger, M. Taqqu, R. Sherman, and D. Wilson. Self-Similarity through high variability: Statistical Analysis of Ethernet LAN traffic at the source level. In *Proceedings of SIGCOMM'95*, 1995.

APPENDIX

$$f(W_m, p, R, B) = \begin{cases} \frac{\frac{1-p}{p} + W(p) + \frac{Q(p, W(p))}{1-p}}{R(W(p)+1) + \frac{Q(p, W(p))G(p)H}{1-p}} & W(p) < W_m \\ \frac{\frac{1-p}{p} + W_m + \frac{Q(p, W_m)}{1-p}}{R(\frac{W_m}{4} + \frac{1-p}{pW_m} + 2) + \frac{Q(p, W_m)G(p)H}{1-p}} & W(p) \geq W_m \end{cases}$$

where:

$$W(p) = \frac{2}{3} + \sqrt{\frac{4(1-p)}{3p} + \frac{4}{9}}$$

$$Q(p, w) = \min \left(1, \frac{(1 - (1-p)^3)(1 + (1-p)^3(1 - (1-p)^{w-3}))}{1 - (1-p)^w} \right)$$

$$G(p) = 1 + p + 2p^2 + 4p^3 + 8p^4 + 16p^5 + 32p^6$$

Multiple Time Scale Redundancy Control for QoS-sensitive Transport of Real-time Traffic

Tsunyi Tuan Kihong Park
Network Systems Lab
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

Abstract:—End-to-end QoS control over best-effort and differentiated service networks which exhibit variability in their exported service properties looms as an important challenge. In previous work, we have shown how packet-level adaptive FEC can be used in dynamic networks to facilitate invariant user-specified QoS in an end-to-end manner.

This paper addresses two important problems—self-similar burstiness and performance degradation of reactive controls subject to long feedback loops—complementing the stability/optimality considerations studied earlier. First, for adaptive redundancy control to be effective, its susceptibility to correlated packet drops and queuing delays stemming from self-similar burstiness must be fortified. Second, to preserve FEC's viability over ARQ when transporting real-time traffic in WANs, proactivity must be injected to offset the performance degradation of reactive feedback controls when subject to long RTTs.

In this paper, we use the recently advanced multiple time scale congestion control framework—first investigated in the throughput maximization context—to endow adaptive redundancy control with both selective protection against self-similar burstiness as well as proactivity to feedback redundancy control. We analyze, implement, and benchmark our protocol—AFEC-MT—in the context of transporting periodic real-time traffic, in particular, MPEG video.

I. INTRODUCTION

A. Background

Forward error correction (FEC) is a well-studied reliable communication technique which has been successfully used, primarily at the *bit-level*, in a number of application domains from space communication to reliable data storage on compact disks [4], [14], [15]. In the context of supporting multimedia traffic with real-time constraints over high-speed wide-area networks, *packet-level* FEC has received interest due to ARQ's inherent limitation at handling timing constraints when subject to long end-to-end latencies [2], [5], [6], [12], [19], [23], [28].

Packet-level FEC introduces further complexities due to correlated packet drops (or erasures) and delays stemming from queueing which is especially severe under self-similar bursty traffic conditions. Cidon et al. [8], [9] have studied the impact of correlated packet drops on packet-level FEC performance and shown that their impact can be significant. Queueing analysis with Poisson input is provided in [1]. Empirical

evidence [5], [23] indicates that performance degradation is further amplified under self-similar traffic conditions.

When applying packet-level FEC for real-time data transport in shared dynamic networks, it is imperative that appropriate redundancy or overcode—commensurate with network state and desired target QoS—is applied such that bandwidth is not unnecessarily wasted. In previous work [22], [23], [26], we proposed an adaptive packet-level FEC protocol called AFEC and analyzed its properties with respect to optimality and stability. The control problem is nontrivial due to the fact that increased redundancy, beyond a certain point, can “backfire” resulting in self-induced congestion which impedes the timely recovery of information at the receiver. We implemented and tested AFEC in high-speed LAN environments when transporting real-time MPEG video and showed that end-to-end QoS provisioning could be facilitated using adaptive redundancy control.

B. Problem Statement

The limitations of our previous work [22], [23] are two-fold: one, AFEC's adaptive redundancy control was geared toward protecting against generic forms of burstiness without special sensitivity to self-similar burstiness [16] thus leaving room for possible improvement, and two, AFEC—being a feedback control—suffered under the problem of long round trip latencies intrinsic to all reactive controls which reduced its effectiveness vis-à-vis ARQ in WAN environments.

We remark that these two problems—although studied in the specific context of adaptive redundancy control for real-time data transport—are also relevant to other forms of end-to-end QoS control over networks exporting variable services [7], [10], [20] where end-to-end control can be used to amplify and endow robustness to the QoS experienced by an application.

C. New Contributions

In this paper we show that the aforementioned problems—self-similar burstiness and feedback redundancy control with long RTTs—can be effectively addressed yielding significant performance improvements. Our solution is based on the framework of *Multiple Time Scale Congestion Control* (MTSC) [29], [30] which has been recently advanced in the

This research was supported by NSF grant ANI-9714707.
T.T.: E-mail: tsunyi@cs.purdue.edu.

K.P.: Contact author; e-mail: park@cs.purdue.edu. Additionally supported by NSF grants ANI-9875789 (CAREER), ESS-9806741, EIA-9972883 and grants from PRF and Sprint.

context of throughput maximization. The basic premise of MTSC hinges on the fact that despite the detrimental performance effect associated with self-similar burstiness the presence of nontrivial correlation structure across multiple time scales admits to exploitation for congestion control purposes. MTSC uses information extracted at large time scales to modulate the output behavior of feedback congestion controls acting at the time scale of the feedback loop. In a nutshell, when the network contention level at large time scales is predicted to be "low," the bandwidth consumption behavior of the underlying feedback congestion control is made more aggressive, and vice versa if the opposite is true.

In the context of adaptive redundancy control for QoS-sensitive transport of real-time traffic, the main technical challenge to adopting the MTSC framework lies in devising a large time scale module which is then coupled to AFEC such that the collective behavior facilitates both selective protection against self-similar burstiness and proactivity to counteract the reactive nature of AFEC. The specific form of coupling in the new protocol—AFEC-MT—is *additive* where the amount of redundancy h applied at an instant is composed of two parts, $h = h_S + h_L$, the short time scale component h_S acting at the time scale of the feedback loop governed by AFEC and the large time scale component h_L . The latter behaves like a "DC component" over the short time scale incurring level shifts at large time scales which reflect the overall contention level and corresponding redundancy needed to achieve a target end-to-end QoS. Whereas h_S is updated using *implicit* prediction afforded by feedback control, h_L is computed using *explicit* prediction.

We give a qualitative analysis of AFEC-MT with respect to its stability and optimality properties. We demonstrate the practical efficacy of the protocol by implementing and benchmarking AFEC-MT when transporting real-time MPEG video over controlled network environments where self-similar burstiness and varying end-to-end latency are systematically injected and their impact evaluated. Of particular interest is the facilitation of proactivity under increasing RTT which mitigates some of the performance cost of reactive controls when subject to broadband networks with a high delay-bandwidth product. We show that significant performance gains are possible by engaging in multiple time scale redundancy control.

The rest of the paper is organized as follows. In the next section, we give an overview of AFEC and its use in real-time communication. This is followed by Section III which describes the MTSC framework. Section IV discusses AFEC-MT and multiple time scale redundancy control, and Section V shows performance results from implementation based experiments for real-time MPEG video transport.

II. OVERVIEW OF AFEC

A. Set-up

Assume k data packets—e.g., representing an MPEG video frame—are encoded as $n = k + h$ packets where $h \geq 0$ is the

degree of redundancy. We will assume that the receipt of *any* k packets out of the n total packets suffices to recover the original k data packets. FEC encoding/decoding functions with this property include Reed-Solomon [18] and IDA [27]. Figure II.1 gives a depiction of packet-level FEC in an end-to-end network environment.

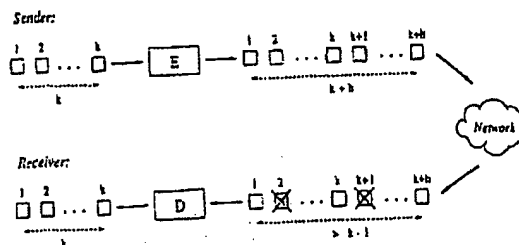


Fig. II.1. A block of k packets encoded at the sender using FEC as $k + h$ packets. If the number of dropped or untimely packets does not exceed h , the original k data packets are recovered at the receiver.

The application in question is *real-time constrained* in the following sense. The sender transmits $n(t_1), n(t_2), \dots, n(t_i), \dots$ ($t_i < t_j$ if $i < j$) blocks of packets at times t_i where $n(t_i) = k(t_i) + h(t_i)$, $i = 1, 2, \dots$. That is, $k(t_i)$ is the traffic requirement at t_i as dictated by the application and $h(t_i)$ the corresponding redundancy factor. We model QoS requirements using *hard* real-time constraints whereby we assume the existence of a monotonically increasing sequence $\{t'_i\}_{i \in \mathbb{Z}^+}$ of deadlines at the receiver such that all $k(t_i)$ data packets belonging to the i 'th block must be recovered before time t'_i . For instance, given a frame rate of 30fps, successive frames must arrive within 33.3ms periodic intervals with some provision for decoding overhead at the receiving end station. QoS is measured at the receiver using a *recovery rate* process $\gamma(t'_i)$ which is defined to be the number of packets belonging to block i received before time t'_i . We will say there is a *hit* at time t'_i if $\gamma(t'_i) \geq k(t_i)$; i.e., decoding of the i 'th frame was timely and successful.

B. Adaptive Redundancy Control

Increasing $h(t)$ blindly will adversely affect $\gamma(t + \tau)$, for some $\tau > 0$. That is, letting $\mathcal{G}, \gamma(t + \tau) = \mathcal{G}(h(t))$, denote the functional relationship between $h(t)$ and $\gamma(t + \tau)$, $\mathcal{G}(h) \geq 0$ is *unimodal* with peak at $h = h^*$. Thus if the goal is to achieve maximum recovery rate, then we arrive at an optimal control problem where the optimal operating point is defined as (h^*, γ^*) , $\gamma^* = \mathcal{G}(h^*)$. If $\gamma^* < k$, then there is a structural problem and no amount of redundancy, small or large, can yield a positive hit rate. Figure II.2 depicts the unimodal redundancy-recovery rate relation $\gamma = \mathcal{G}(h)$.

Consider the case when $\gamma^* \geq k$. Assuming $\gamma(t)$ is fed back to the sender from the receiver, we can formulate the following

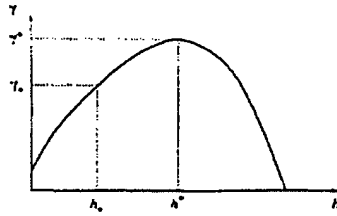


Fig. 11.2. Unimodal redundancy-recovery rate function $\gamma = G(h)$ with maximum recovery rate γ^* and target recovery rate γ_* .

control law

$$\frac{dh(t)}{dt} = \epsilon(\gamma_* - \gamma(t - \tau)) \quad (11.1)$$

where γ_* , $k \leq \gamma_* \leq \gamma^*$, is the target recovery rate, $\epsilon > 0$ is an adjustment parameter, and $\tau \geq 0$ is a delay term introduced by feedback and network latency. The control algorithm as embodied by (11.1) just says that if the measured recovery rate $\gamma(t - \tau)$ is smaller than the target recovery rate γ_* , then the redundancy factor h should be increased, and vice versa.

Instability ensues when "too much" redundancy is applied at the sender which is tantamount to shooting oneself in the foot. Let

$$\begin{aligned} H_L &= \{(h, \gamma) : h = G(h), h < h^*\}, \\ H_R &= \{(h, \gamma) : h = G(h), h \geq h^*\}. \end{aligned}$$

It can be shown [22], [25] that target operating points belonging to $(h_*, \gamma_*) \in H_L$ are asymptotically stable whereas those belonging to $(h_*, \gamma_*) \in H_R$ are unstable. To achieve stability, we augment the control law given in (11.1) via a *directional check* given by the sign of $d\gamma/dh$. If $d\gamma/dh > 0$, then the system is in the stable region ($h < h^*$) and (11.1) is applied as usual. If, on the other hand, $d\gamma/dh \leq 0$, then the system finds itself in the unstable region ($h \geq h^*$) and a *backoff mechanism*— $dh/dt < 0$ —is instituted until $d\gamma/dh > 0$ at which time we find ourselves again in the stable regime. Note that $d\gamma/dh > 0$ iff $(h, \gamma) \in H_L$.

The augmented AFEC algorithm containing both the symmetric and asymmetric control components is given by

$$\frac{dh(t)}{dt} = \begin{cases} \epsilon(\gamma_* - \gamma(t - \tau)), & \text{if } d\gamma(t - \tau)/dh > 0, \\ -ah, & \text{otherwise.} \end{cases}$$

Here $a > 0$ is a positive constant. Hence the augmented control follows (11.1), as it should, when $h < h^*$ (i.e., $(h, \gamma) \in H_L$), and it performs drastic, asymmetric backoff only when $h \geq h^*$ ($(h, \gamma) \in H_R$). The backoff mechanism, $dh/dt = -ah$, is exponential leading to a decay of h of the form e^{-at} .

III. OVERVIEW OF MTSC

A. Self-Similar Burstiness

Let $\{X_i; i \in \mathbb{Z}_+\}$ be a time series which represents the trace of data traffic measured at some fixed time granularity. We

define the aggregated series $X_i^{(m)}$ as

$$X_i^{(m)} = \frac{1}{m}(X_{im-m+1} + \dots + X_{im}).$$

That is, X_i is partitioned into blocks of size m , their values are averaged, and i is used to index these blocks. Let $r(k)$ and $r^{(m)}(k)$ denote the autocorrelation functions of X_i and $X_i^{(m)}$, respectively. Assume X_i has finite mean and variance. X_i is *asymptotically second-order self-similar* with parameter H ($1/2 < H < 1$) if for all $k \geq 1$,

$$r^{(m)}(k) \sim \frac{1}{2}((k+1)^{2H} - 2k^{2H} + (k-1)^{2H}) \quad (11.1)$$

as $m \rightarrow \infty$. H is called the *Hurst parameter* and its range $1/2 < H < 1$ plays a crucial role. The significance of (11.1) stems from the following two properties being satisfied:

(i) $r^{(m)}(k) \sim r(k)$,

(ii) $r(k) \sim ck^{-\beta}$,

as $k \rightarrow \infty$ where $0 < \beta < 1$ and $c > 0$ is a constant. Property (i) states that the correlation structure is preserved with respect to time aggregation, and it is in this second-order sense that X_i is "self-similar." Property (ii) says that $r(k)$ behaves hyperbolically which implies $\sum_{k=0}^{\infty} r(k) = \infty$. This is referred to as *long-range dependence* (LRD). The second property hinges on the assumption that $1/2 < H < 1$ as $H = 1 - \beta/2$.

The relevance of asymptotic second-order self-similarity for network traffic derives from the fact that it plays the role of a "canonical" model where the on/off model of Willinger *et al.* [31], Likhanov *et al.*'s source model [17], and the M/G/∞ queueing model with heavy-tailed service times [11]—among others—all lead to second-order self-similarity. In general, self-similarity and long-range dependence are not equivalent. For example, fractional Brownian motion with $H = 1/2$ is self-similar but it is not long-range dependent. For second-order self-similarity, however, one implies the other and it is for this reason that we sometimes use the terms interchangeably within the traffic modeling context. A more comprehensive discussion can be found in [24].

B. LRD and Predictability

Given X_i and $X_i^{(m)}$, we will be interested in estimating $\Pr\{X_{i+1}^{(m)} | X_i^{(m)}\}$ for some suitable aggregation level $m > 1$. If X_i is short-range dependent, we have

$$\Pr\{X_{i+1}^{(m)} | X_i^{(m)}\} \sim \Pr\{X_{i+1}^{(m)}\}$$

for large m whereas for long-range dependent traffic, correlation provided by conditioning is preserved. Thus given traffic observations $a, b > 0$ ($a \neq b$) of the "recent" past corresponding to time scale m ,

$$\Pr\{X_{i+1}^{(m)} | X_i^{(m)} = b\} \neq \Pr\{X_{i+1}^{(m)} | X_i^{(m)} = a\}$$

¹That is, via its relation to fractional Brownian motion and its increment process, fractional Gaussian noise.

and this information may be exploited to enhance congestion control actions undertaken at smaller time scales. We employ a simple, easy-to-implement—both on-line and off-line—prediction scheme to estimate $\Pr\{X_{i+1}^{(m)} | X_i^{(m)}\}$ based on observed empirical distribution. We note that *optimum estimation* is a difficult problem for LRD traffic [3], and its solution is outside the scope of this paper. Our estimation scheme provides sufficient accuracy with respect to extracting predictability and is computationally efficient, however, it can be substituted by any other scheme if the latter is deemed “superior” without affecting the conclusions of our results. To facilitate normalized contention levels, we define a map $L: \mathbb{R}_+ \rightarrow [1, s]$, monotone in its argument, and let $x_i^{(m)} = L(X_i^{(m)})$. Thus $x_i^{(m)} \approx 1$ is interpreted as the aggregate traffic level at time scale m being “low” and $L_s \approx s$ is understood as the traffic level being “high.” The process $x_i^{(m)}$ is related to the *level process* used in [13] for modeling LRD traffic. We use L_1 and L_2 without reference to the specific time index i to denote consecutive quantized traffic levels $x_i^{(m)}, x_{i+1}^{(m)}$.

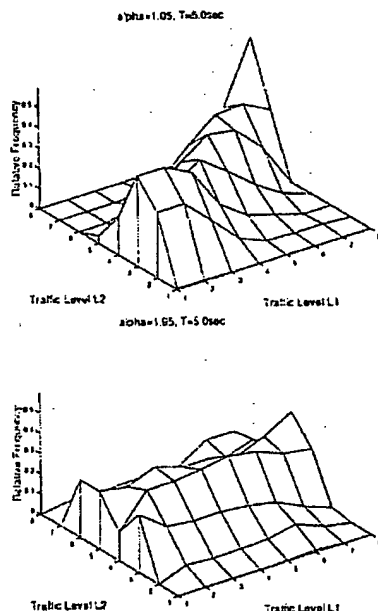


Fig. III.1. Conditional probability densities with L_2 conditioned on L_1 for LRD traffic (top) and SRD traffic (bottom).

Figure III.1 (top) shows the predictability structure of LRD traffic at a time scale of 5s by plotting its 3-D conditional probability densities. The diagonal skewness indicates that conditioning on L_1 is informative with respect to predicting L_2 . Figure III.1 (bottom) shows the corresponding densities for short-range dependent traffic. We observe that conditioning has negligible influence.

C. Coupling

Multiple time scale congestion control allows for n -level time scale congestion control ($n \geq 1$) where information extracted at n separate time scales $T_1 < T_2 < \dots < T_n$ is cooperatively engaged to modulate the output behavior of the feedback congestion control residing at the lowest time scale (i.e., $n = 1$). The objective of MTSC is to improve performance vis-à-vis the congestion control consisting of the feedback congestion control alone. We concentrate on 2-time scale congestion control where the “large” time scale module C_L —separated by an order of magnitude from the “small” time scale module C_S —is coupled to the latter to yield a new control C_{LOS} . For throughput maximization, for example, the coupling takes on a *multiplicative* form [29]. For QoS control us-

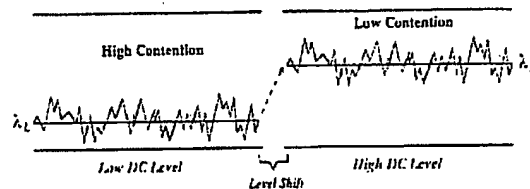


Fig. III.2. Additive coupling via selective “DC” level adjustment—i.e., level shift—between high- and low-contention periods.

ing adaptive FEC, we employ *additive* coupling. The latter is illustrated in Figure III.2 where a “DC” level shift is instituted with respect to the large time scale rate which results in an increase of the base rate from λ_L to λ_H .

IV. AFEC-MT

A. Multiple Time Scale Redundancy Control

AFEC-MT, in general, allows for n -level time scale redundancy control for $n \geq 1$ where information extracted at n separate time scales is additively coupled to yield the level of redundancy h applied at the FEC encoder. This is depicted in Figure IV.1.

Our design methodology is based on devising the large time scale module at time scale T_2 , attaching it to the AFEC module operating at time scale T_1 of the feedback loop to improve the control actions undertaken by AFEC. Our objective is to show that this modular extension results in a control which is able to achieve significant performance gains relative to AFEC. In the 2-time scale redundancy control setting, the explicit prediction component of the large time scale module C_L outputs a redundancy level $h_L (\equiv h_2)$ which is then *additively* combined with the redundancy level $h_S (\equiv h_1)$ computed by C_S , i.e., AFEC. $h = h_S + h_L$ is then passed on to the FEC encoder component of AFEC. In specifying the control law

$$\frac{dh}{dt} = \frac{dh_S}{dt} + \frac{dh_L}{dt}$$

the control governing h_S is AFEC (cf. Section II-B) hence needs no separate description.

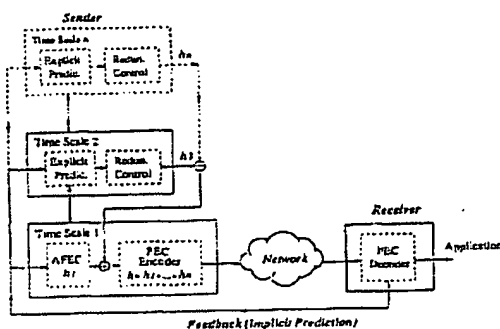


Fig. IV.1. AFEC-MT framework. Dashed lines show the potential extensibility of the framework to three or more time scales.

B. Structure of C_L

Whereas dh_S/dt is affected at the time scale T_1 of AFEC's feedback loop, dh_L/dt is affected at the much larger time scale T_2 . This implies that the frequency of updates for h_S are much greater than that of h_L . Now to the description of dh_L/dt .

B.1 Explicit Prediction

The explicit prediction component of C_L performs per-connection on-line estimation of the conditional probability densities $\Pr\{L_2 | L_1 = k\}$, $k \in [1, s]$, following the method outlined in Section III-B. It turns out that on-line estimation can be accomplished using $O(1)$ operations at every update interval, i.e., C_L 's time scale T_2 . On the sender side, C_L maintains a 2-dimensional array $\text{CondProb}[\cdot][\cdot]$ of size $s \times (s+1)$, one row for each $k \in [1, s]$. The last column of CondProb , $\text{CondProb}[k][s+1]$, is used to keep track of the number of blocks observed thus far whose traffic level map to k . Since $\Pr\{L_2 = \ell | L_1 = k\} = \text{CondProb}[k][\ell] / \text{CondProb}[k][s+1]$, having the table CondProb means having the conditional probability densities.

B.2 Selective Redundancy Control

Using the conditional probability density table CondProb , we compute the expectation of L_2 conditioned on L_1 , $\ell = E(L_2 | L_1 = k)$, $\ell \in [1, s]$, at the end of each L_1 . ℓ is then used to index the redundancy schedule $H : [1, s] \rightarrow \mathbb{R}_+$ to yield the value

$$h_L := H(\ell).$$

What remains is the computation of the function H . Let h^1, h^2, \dots, h^s denote the function values of H . Each component h^ℓ is updated according to the symmetric control

$$\frac{dh^\ell}{dt} = \begin{cases} \nu, & \text{if } d\gamma/dh^\ell > 0 \text{ and } \gamma_* > \gamma, \\ -\nu, & \text{if } d\gamma/dh^\ell < 0 \text{ or } \gamma - \gamma_* > \theta, \end{cases}$$

where $\nu > 0$ is an adjustment factor and $\theta \geq 0$ is a threshold parameter. The sign of $d\gamma/dh^\ell$ can be estimated by maintain-

ing a history of redundancy action-QoS impact pair sequences, one for each contention level $\ell \in [1, s]$. At time $t = 0$, the initial values of H are set to zero. The value of h^ℓ , $\ell \in [1, s]$, is updated at the end of each block L_1 whose conditional expectation maps to ℓ .

V. REAL-TIME MPEG VIDEO TRANSPORT

A. System Structure

We have built an implementation of AFEC-MT customized for the transport of real-time MPEG video. For brevity, we will refer to this system as AFEC-MT in the following sections. AFEC-MT consists of a number of modules including the FEC codec, receiver-side controller C_R , and sender-side controllers C_S^1 and C_S^2 . C_S^2 adjusts short-range redundancy h_1 by reacting to feedback at the time scale of RTT. C_S^2 is implemented "on top" of C_S^1 via the coupling described in Section IV and sets the long-range redundancy level h_2 . The net redundancy $h = [h_1 + h_2]^+$ is then input to the FEC encoder. The system can be configured to turn off either one of the two control modules. Thus when C_S^2 is disabled, the system degenerates to AFEC. On the sender side, a stream of I , P , B frames is generated by an MPEG encoder at some frame rate f (e.g., 25–30 frames/sec). The FEC encoder applies forward error correction on each frame producing a sequence of $n = k + h$ packets which are submitted to the network.

At the receiver, upon receiving a sequence of packets belonging to the same frame, the controller C_R checks if at least k packets have arrived in a timely manner. If the number of timely packets is less than k , then the packets belonging to the current frame are discarded including those arriving subsequently. If at least k packets arrived within their deadline, then the first k packets (any k -subset will do) are forwarded to the FEC decoder proper which decodes the packet stream to recover the original k data packets constituting the sender's frame. The I , P , or B frame is then forwarded to the MPEG II player which applies its own decoding to produce the uncompressed frame which is then rendered on the terminal as part of the video stream. We employ Rabin's IDA [27] as the

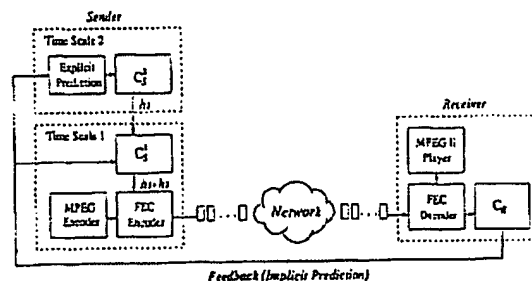


Fig. V.1. System structure of AFEC-MT.

FEC codec. Details of the FEC implementation can be found in [22], [23]. C_R computes certain control information including

the number of timely arrived packets γ which is then fed back via a control packet to the sender. All of the modules are implemented in software, and it is completely end-to-end such that the system can be deployed over any IP network. The forward flow structure is shown in Figure V.1.

B. Experimental Set-up

B.1 Network Configuration

Our experiments were carried out in the test environment shown in Figure V.2. AFEC-MT, consisting of the AFEC-MT sender and receiver, sent its traffic over a router where cross traffic stemming from a separate cross traffic source was multiplexed with the application traffic. By varying the cross traffic characteristics as well as the resources (e.g., buffer capacity) at the router, a wide spectrum of contention levels could be produced. To facilitate a controlled environment for measurement purposes, the camera and MPEG I encoder at the sender were replaced by an emulator that fed the frames of stored MPEG I video at real-time frame rates to the AFEC-MT sender. Thus as far as AFEC-MT was concerned, its functioning was not affected since—in either case— I, P, B frames were input to the FEC encoder at real-time speeds². The MPEG II player at the receiver was always run, and the performance measurements reflect the overhead incurred by the player and FEC decoding processing overhead. Thus modulo where the I, P, B frames came from at the sender-side, the system depicted in Figure V.1 was faithfully implemented with all the components implemented in software without specialized hardware support.

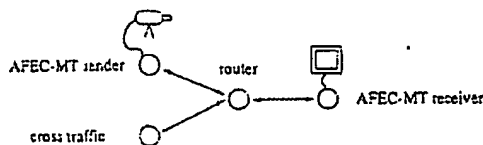


Fig. V.2. Experimental set-up for AFEC-MT performance measurements.

The topology depicted in Figure V.2 was realized over a private FastEthernet LAN environment with the AFEC-MT sender/receiver, cross traffic source, and router running on four Sun UltraSparc 1 & 2 workstations. Without the configurable router, we found that the 100 Mbps bandwidth of FastEthernet was too large relative to the data rate of MPEG I video (~1.5 Mbps), even with our cross traffic source active, to cause significant contention. With an UltraSparc 2 node acting as a configurable router implementing FIFO packet scheduling, a wide range of contention levels could be created under controlled conditions. The AFEC-MT protocol as well as the cross traffic source ran on top of UDP. AFEC-MT was realized as an application layer process portable to other UNIX environments.

²An implementation of the AFEC-MT sender which interfaces with an Optibase real-time MPEG I compression engine is available for Windows NT.

B.2 Benchmark Traffic

Self-similar cross traffic was generated by utilizing traces from [21]. We emphasize that the performance results are obtained using actual MPEG I video rather than frame size traces commonly used in simulation and experimentation set-ups. Thus the cost of FEC encoding and decoding, control actions, and MPEG II player's processing overhead are all reflected in the performance results. We use the MPEG I video clip, Beauty and the Beast, as the main benchmark data source. The clip consists of 36000 frames which, at 15 f/s frame rate, has a duration of 40 minutes. Other real-time video data employed include Simpsons cartoon and Terminator clips.

C. Performance Measurement

C.1 Unimodal Redundancy-Recovery Relation

Figure V.3 (top) shows the measured redundancy-recovery function for the Beauty and the Beast MPEG I video clip transmission when using static FEC with h fixed in the range 0–14. We observe that the redundancy-recovery function is unimodal with peak at $h = 8$. Figure V.3 (bottom) shows the correspond-

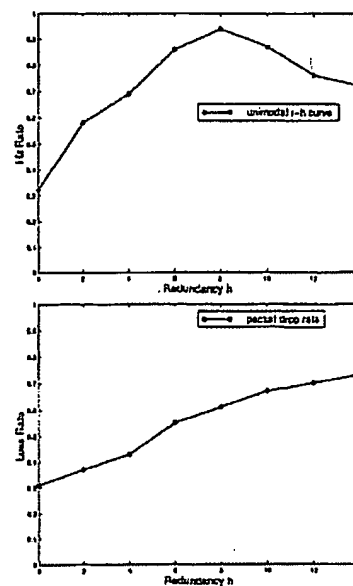


Fig. V.3. Top: Measured unimodal redundancy-recovery function for Beauty and the Beast MPEG I video clip; Bottom: Corresponding packet loss rate function.

ing packet loss rate curve which monotonically increases with h . The increase in packet loss rate stems from the higher traffic rate associated with increased redundancy.

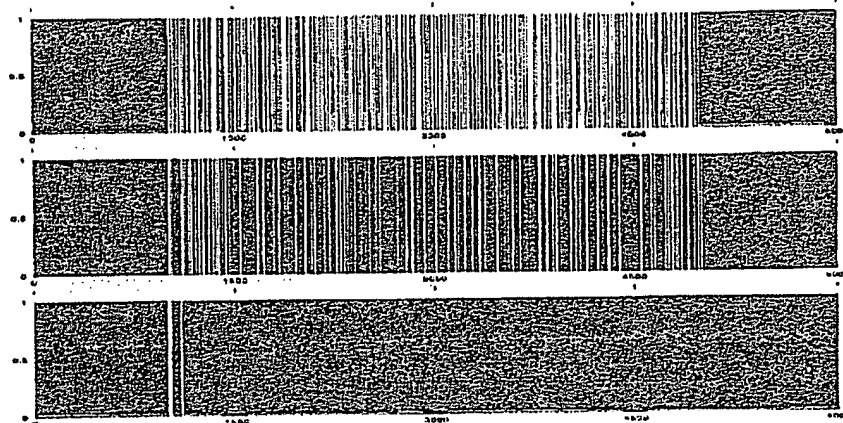


Fig. V.4. Hit trace for static FEC (top), AFEC (middle), and AFEC-MT (bottom) for self-similar cross traffic active during the middle time interval [1000,5000], silent otherwise.

C.2 AFEC-MT vs. AFEC

The dynamical performance of AFEC-MT vis-à-vis AFEC and static FEC can be gleaned from Figure V.4. The figures show impulse plots over time—represented as frame sequence numbers—where the presence of a unit impulse indicates that the corresponding frame was decoded timely at the receiver. The absence of unit impulse (i.e., white stripe) shows frames which did not meet their deadline. Figure V.4 (top) shows performance of static FEC with $h = 0$ when a self-similar cross traffic source was active during the middle time interval [1000,5000] while being silent otherwise. Interference of cross traffic sharing a FIFO queue at the router degrades performance of the application flow which manifests itself as degraded QoS—i.e., hit rate—during the middle interval. Figure V.4 (middle) shows corresponding performance of AFEC for the same set-up and cross traffic conditions. We observe that performance is significantly improved as shown by the reduction in missed deadlines which translates to improved end-to-end QoS as perceived by the user. Figure V.4 (bottom) shows performance of AFEC-MT which further improves upon AFEC's achieved QoS yielding a measured hit rate that is close to the user's desired hit rate. We observe a small interval of timeliness violation at time 1000—the onset of cross traffic—which is a nonstationary, unpredictable event to which AFEC-MT then adjusts subsequently.

Whereas Figure V.4 depicts “instantaneous” QoS, Figure V.5 shows the running average of measured hit rate at the receiver for a set-up involving 12000 frames during which the self-similar cross traffic was always active. Figure V.5 (left) shows mean hit rate for static FEC with $h = 0$ whose hit rate at 0.21 is significantly below the target hit rate of 0.92. The middle figure shows corresponding performance of AFEC which improves upon the performance of static FEC but is still notice-

ably below the user-specified target level. Figure V.5 (right) shows performance of AFEC-MT which converges to the target hit rate after a transient initial adjustment period. The target hit rate is reached much faster than indicated in the plot: the latter shows the long-term running average of instantaneous hit rates from time 0 onwards, not a local window.

We remark that the improvement of AFEC-MT over AFEC stems from multiple time scale AFEC's ability to more accurately discern short-term fluctuations from persistent changes which allows it to be less reactive to short-range variations. AFEC's advantage over static FEC lies in its ability to tailor redundancy to current network state, increasing it when needed to shield QoS, and decreasing redundancy when not needed to reduce wastage of shared network resources. This trade-off between QoS and bandwidth imparts a cost for reducing redundancy in response to short-term changes in network state as, in the near future, a QoS violation is likely to arise due to reduced protection which will then subsequently trigger an increase in redundancy. Bandwidth may have been temporarily “saved” by decreasing redundancy in response to short-term changes, but given the *primary goal* of achieving a target QoS while minimizing bandwidth usage in so doing—a *secondary objective*—responsiveness to short-term fluctuations is undesirable. Thus AFEC-MT's ability to discriminate and undertake differentiated action with respect to redundancy in the form of a short-range component h_1 and long-range component h_2 endows it with enhanced prowess to deliver end-to-end QoS while being efficient with respect to its resource usage.

C.3 RTT and Proactivity

As the round trip time (RTT) associated with the feedback loop increases, the state information conveyed by feedback becomes more outdated, and the effectiveness of reactive actions undertaken by a feedback control diminishes. The penalty is

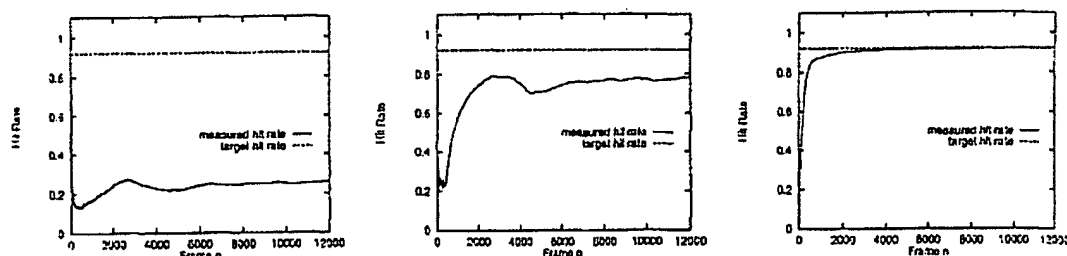


Fig. V.5. QoS performance comparison: static FEC (left), AFEC (middle), and AFEC-MT (right).

amplified in broadband wide area networks where the delay-bandwidth product increases proportionally with delay or bandwidth. Figure V.6 (left) shows measured hit rate as a function of RTT for both AFEC-MT and AFEC. We observe that AFEC's hit rate decreases significantly as RTT is increased which is commensurate with the effect of outdatedness of feedback information. AFEC-MT, on the other hand, suffers significantly less under the same conditions maintaining a much flatter performance curve. Thus AFEC-MT is able to mitigate part of the cost incurred by reactive control by exploiting long-range correlation structure to reduce the performance impact of outdated feedback.

Figure V.6 (right) shows the relative performance gain of AFEC-MT vis-à-vis AFEC where *performance gain* ν is defined as

$$\nu = \frac{\gamma_{AFEC-MT} - \gamma_{AFEC}}{\gamma_{AFEC}}$$

Assuming $\gamma_{AFEC-MT} \geq \gamma_{AFEC}$, $\nu \geq 0$ represents the percentage of improvement achieved by AFEC-MT over AFEC. We observe that ν increases with RTT indicating that AFEC's susceptibility to long round-trip times increases vis-à-vis the corresponding susceptibility of AFEC-MT.

C.4 Impact of Long-range Dependence

We have shown that the correlation structure in self-similar traffic—upon effective utilization by AFEC-MT—leads to enhanced QoS above and beyond what AFEC can provide. Yet another dimension of interest is the impact of long-range correlation structure—i.e., its strength—on performance gain. Table I shows the hit rate of AFEC, AFEC-MT, and performance gain when the long-range dependence present of cross traffic is increased from weak ($\alpha = 1.95$) to strong ($\alpha = 1.05$). Network traffic measurements correspond to traffic with $\alpha \approx 1$. The α measure is related to the Hurst parameter for measuring long-range dependence, and we refer the reader to [21], [30] for a more detailed discussion.

Table I shows that performance gain amplifies as long-range dependence is increased with α approaching 1. Thus at the same time that long-range dependence exerts a negative influence on performance from the queuing perspective, the same

structure can be exploited to affect control decisions that mitigate the very performance effects that are caused by long-range correlations in the first place. This is the "good news within the bad news" syndrome [29]. We remark that when varying the long-range dependence associated with cross traffic, it is important that all other things are kept equal—including the average traffic rate—to preserve comparability. Generating traffic loads with this property is a nontrivial task due to sampling error introduced when engaging heavy-tailed distributions in physical traffic models. Details for generating normalized workloads is provided in [30].

TABLE I
IMPACT OF LONG-RANGE DEPENDENCE ON PERFORMANCE GAIN OF
AFEC-MT VS. AFEC.

α	1.05	1.35	1.65	1.95
AFEC	0.764	0.831	0.882	0.920
AFEC-MT	0.919	0.918	0.921	0.917
Gain	20.3%	10.47%	4.42%	-0.0%

Table I also shows that when traffic is weakly correlated at large time scales ($\alpha = 1.95$), the performance difference between AFEC-MT and AFEC is minimal (0.917 vs. 0.920). This is not surprising since given that there is little structure at large time scales to exploit, the performance benefit ensuing from coupling AFEC with the large time scale control module should be commensurately low or nonexistent. At the very least, AFEC-MT should not "hurt" performance vis-à-vis AFEC which we find is the case.

D. Redundancy Schedule

Related to the impact of long-range dependence is the time evolution of the redundancy schedule $H(\cdot)$ which for predicted traffic level ℓ at time scale T_2 assigns the base redundancy $h_L = H(\ell)$. The update of the components of $H(\ell)$, $\ell = 1, 2, \dots, 8$, is affected by the symmetric control law given in Section IV-B.2. Figure V.7 (top) shows the evolution of $H(\cdot)$ as a function of time (represented as frame numbers here) for $\alpha = 1.05$ traffic. Figure V.7 (bottom) shows the corresponding values for $\alpha = 1.95$ traffic. We observe that when large time

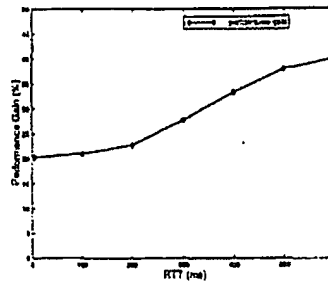
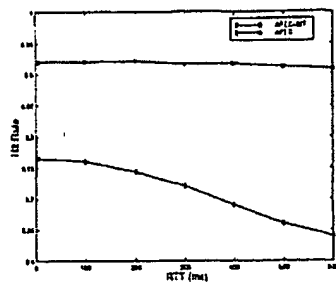


Fig. V.6. Proactivity of AFEC-MT as a function of RTT. Left: Hit rate of AFEC-MT vs. hit rate of AFEC. Right: Performance gain of AFEC-MT relative to AFEC.

scale correlation structure is weak, then the values are concentrated around a narrow range (i.e., 3–5) which points toward the fact that conditioning on predicted large time scale traffic level is of limited utility. For $\alpha = 1.05$ traffic, on the other hand, the redundancy values are spread out over a much wider range (i.e., 0–8) which indicates that conditioning does provide the ability to discriminate with respect to the future. In particular, this allows a base redundancy of $h_L = 8$ to be applied when traffic contention is high ($\ell = 8$) to facilitate increased protection, and a correspondingly small “DC” redundancy $h_L = 0$ when persistent traffic contention is low ($\ell = 1$).

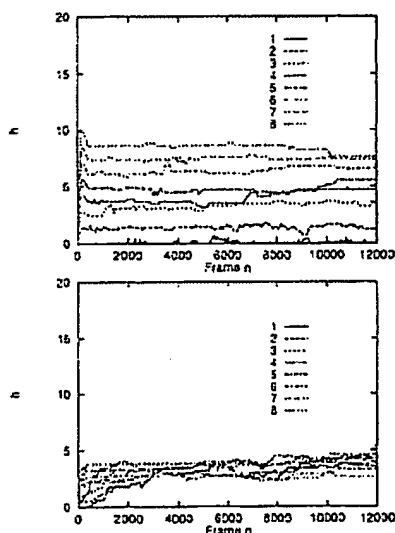


Fig. V.7. Evolution of redundancy schedule as a function of time. Top: $\alpha = 1.05$ traffic. Bottom: $\alpha = 1.95$ traffic.

D.1 Multiple AFEC-MT Connections

AFEC-MT is an end-to-end protocol designed to run in shared network environments where multiple AFEC-MT flows

compete for available resources. As seen in the control laws for AFEC and its large time scale module that adjusts the redundancy schedule (cf. Sections IV-B.2 and II-B), AFEC-MT tries to achieve a target QoS—i.e., hit rate—while applying an amount of redundancy deemed adequate to do so, but not more. In this sense, it is a cooperative protocol, just like TCP, which stands in contrast to noncooperative protocols that would exploit other flows’ cooperativeness and not back off even under adverse conditions³, thereby absorbing their bandwidth [7]. In fact, if the user-specified target hit rate is sufficiently low, then AFEC-MT—when compared to TCP—may end up consuming less bandwidth than TCP which tries to maximize throughput. Of course, AFEC-MT can be transformed into a protocol which seeks to maximize QoS, in which case, its modus operandi is analogous to that of TCP: back-off is instituted only when a further increase in redundancy would result in a decreased hit rate.

Table II shows hit rate for a network environment consisting of three AFEC-MT connections. The three connections compete for resources at the same bottleneck router shown in Figure V.2 as for the single AFEC-MT connection case. The cross traffic course remains the same. Table II shows bandwidth sharing behavior for three network conditions—when the cross traffic source is sending at a rate of 6.4Mbps (network contention is high), 4.8Mbps, and 3.2Mbps (network contention is low). The target hit rate for each connection was set at 0.92. When available bandwidth is small (first column), network resources are insufficient to achieve the target hit rate 0.92. The flows remain stable due to the back-off mechanism of AFEC-MT, and all three connections end up sharing an approximately equal amount of bandwidth which results in similar hit rates of 0.56, 0.53, and 0.57, respectively. Note that this is equivalent to a “maximize QoS” mode. Bandwidth sharing behavior stays invariant as the traffic rate of the cross traffic source is decreased, eventually leading to all three connections achieving their target hit rate 0.92 (the second connection is a fraction short) when network contention is low (last column). Performance evalua-

³ That is, network state where increasing redundancy leads to a decrease in hit rate for the flow in question.

tion with four or more connections were carried out using simulations yielding qualitatively similar results.

TABLE II
QOS PERFORMANCE OF MULTIPLE AFEC-MT CONNECTIONS WITH
RESPECT TO FAIRNESS.

Cross Traffic	6.4 Mb/s	4.8 Mb/s	3.2 Mb/s
Connection I	0.56	0.82	0.92
Connection II	0.53	0.83	0.91
Connection III	0.57	0.79	0.92

VI. CONCLUSION

In this paper, we have introduced a multiple time scale extension of AFEC—a protocol that performs packet-level adaptive FEC for real-time payload transport [22], [23], [26]. A large time scale module that extracts long-range correlation structure in network contention was constructed and coupled with AFEC yielding its extension AFEC-MT. The large time scale module augmented the feedback redundancy control mechanism employed by AFEC by additively engaging a "DC" redundancy level as a function of predicted, long-range network state. We implemented AFEC-MT for real-time MPEG video transport and showed that significant performance gains are achievable by utilizing long-range correlation structure present in self-similar traffic. An important consequence of AFEC-MT is its ability to impart proactivity when subject to long round-trip times. By exploiting long-range correlation structure at time scales an order of magnitude higher than the RTT of the feedback loop, the detrimental performance effect of outdated feedback information is mitigated, which is especially severe in broadband wide area networks that possess a high delay-bandwidth product.

REFERENCES

- [1] O. Alt-Hellal, E. Altman, A. Jean-Marie, and I. Kurkova. On loss probabilities in presence of redundant packets and several traffic sources. *Performance Evaluation*, 36:485-513, 1999.
- [2] A. Albanese, J. Blomer, J. Edmonds, M. Luby, and M. Sudan. Priority encoding transmission. *IEEE Trans. Information Theory*, 42(6):1737-1744, 1995.
- [3] Jan Beran. *Statistics for Long-Memory Processes*. Monographs on Statistics and Applied Probability. Chapman and Hall, New York, NY, 1994.
- [4] E. Berlekamp, R. Peile, and S. Pope. The application of error control in communications. *IEEE Communications Magazine*, 25(4):44-57, 1987.
- [5] Ernst Biersack. Performance evaluation of forward error correction in ATM networks. In *Proc. ACM SIGCOMM '92*, pages 248-257, 1992.
- [6] J. Bolot, S. Fosse-Parisis, and D. Towsley. Adaptive FEC-based error control for Internet telephony. In *Proc. IEEE INFOCOM '99*, pages 1453-1460, 1999.
- [7] S. Chen and K. Park. An architecture for noncooperative QoS provision in many-switch systems. In *Proc. IEEE INFOCOM '99*, pages 864-872, 1999.
- [8] I. Cidon, A. Khamisy, and M. Sidi. Analysis of packet loss processes in high-speed networks. *IEEE Trans. Information Theory*, 39(1):98-103, 1993.
- [9] I. Cidon, A. Khamisy, and M. Sidi. Delay, jitter and threshold crossing in ATM systems with dispersed messages. *Performance Evaluation*, 29(2):85-104, 1997.
- [10] D. Clark and W. Fang. Explicit allocation of best-effort packet delivery service. *IEEE/ACM Trans. Networking*, 6(4):362-373, 1998.
- [11] D. R. Cox. Long-range dependence: a review. In H. A. David and H. T. David, editors, *Statistics: An Appraisal*, pages 55-74. Iowa State Univ. Press, 1984.
- [12] S. Dravida and R. Damodaram. Error detection and correction options for data services in B-ISDN. *IEEE J. Select. Areas Commun.*, 9(9):1482-1495, 1991.
- [13] N. Duffield and W. Whitt. Network design and control using on-off and multi-level source traffic models with heavy-tailed distributions. In K. Park and W. Willinger, editors, *Self-Similar Network Traffic and Performance Evaluation*. Wiley Interscience, 1999.
- [14] S. Kaneda and E. Fujiwara. Single bit error correcting double bit error detecting code for memory systems. *IEEE Trans. on Computers*, C-31(5):596-602, 1982.
- [15] K. Kawashima and H. Saito. Teletraffic issues in ATM networks. *Computer Networks and ISDN Systems*, 20:369-375, 1990.
- [16] W. Leland, M. Taqqu, W. Willinger, and D. Wilson. On the self-similar nature of ethernet traffic. In *Proc. ACM SIGCOMM '93*, pages 183-193, 1993.
- [17] N. Likhonov, B. Tsybakov, and N. Georganas. Analysis of an ATM buffer with self-similar ("fractal") input traffic. In *Proc. IEEE INFOCOM '95*, pages 985-992, 1995.
- [18] F. MacWilliams and N. Sloane. *The Theory of Error Correcting Codes*. North-Holland, New York, 1977.
- [19] Anthony McAuley. Reliable broadband communication using a burst erasure correcting code. In *Proc. ACM SIGCOMM '90*, pages 197-206, 1990.
- [20] K. Nichols, V. Jacobson, and L. Zhang. A two-bit differentiated services architecture for the Internet. Internet Draft, 1997.
- [21] K. Park, G. Kim, and M. Crovella. On the relationship between file sizes, transport protocols, and self-similar network traffic. In *Proc. IEEE International Conference on Network Protocols*, pages 171-180, 1996.
- [22] K. Park and W. Wang. AFEC: an adaptive forward error correction protocol for end-to-end transport of real-time traffic. In *Proc. IEEE ICIN*, pages 196-205, 1998.
- [23] K. Park and W. Wang. QoS-sensitive transport of real-time MPEG video using adaptive forward error correction. In *Proc. IEEE Multimedia Systems '99*, pages 426-432, 1999.
- [24] K. Park and W. Willinger. Self-similar network traffic: An overview. In K. Park and W. Willinger, editors, *Self-Similar Network Traffic and Performance Evaluation*. Wiley Interscience, 1999.
- [25] Kihong Park. Warp control: a dynamically stable congestion protocol and its analysis. In *Proc. ACM SIGCOMM '93*, pages 137-147, 1993.
- [26] Kihong Park. AFEC: an adaptive forward error-correction protocol and its analysis. Technical Report CSD-TR-97-038, Department of Computer Sciences, Purdue University, 1997.
- [27] Michael Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM*, 36(2):335-348, 1989.
- [28] N. Shacham and P. McKenny. Packet recovery in high-speed networks using coding. In *Proc. IEEE INFOCOM '90*, pages 124-131, 1990.
- [29] T. Tuan and K. Park. Multiple time scale congestion control for self-similar network traffic. *Performance Evaluation*, 36:359-386, 1999.
- [30] T. Tuan and K. Park. Performance evaluation of multiple time scale TCP under self-similar traffic conditions. Technical Report CSD-TR-99-0-0, Department of Computer Sciences, Purdue University, 1999.
- [31] W. Willinger, M. Taqqu, R. Sherman, and D. Wilson. Self-similarity through high-variability: statistical analysis of Ethernet LAN traffic at the source level. In *Proc. ACM SIGCOMM '95*, pages 100-113, 1995.

HPF: A Transport Protocol For Supporting Heterogeneous Packet Flows in the Internet

Jia-Ru Li Sungwon Ha Vaduvur Bharghavan

Coordinated Sciences Laboratory

University of Illinois at Urbana-Champaign

Email: {juru, s-ha, bharghav}@timely.crhc.uiuc.edu

Abstract:—We present a new transport protocol called HPF for effectively supporting heterogeneous packet flows in the Internet environment. The following are the key features of HPF:

- HPF supports packet flows where different packets in the same transport connection have different quality-of-service requirements in terms of reliability, priority, and deadlines.
- HPF supports application-level framing, and provides APIs for applications to specify the priority, reliability and timing requirements of each frame.
- HPF enables the use of application-specified priorities as hints for network routers to preferentially drop low-priority packets during congestion. This ensures that 'important data' gets through preferentially during congestion.
- HPF decouples the congestion control and reliability mechanisms in order to support congestion control for unreliable and heterogeneous packet flows.

Preliminary performance measurements in our experimental testbed show that HPF can provide effective support for heterogeneous packet flows in the presence of dynamic network resources.

1. INTRODUCTION

The explosion in the use of the Internet in recent years has exposed several limitations in its design. Until recently, user traffic has been predominantly data-oriented, resulting in transport protocols that have focused on providing reliable data transport and congestion control for flows in which all packets have the same requirements in terms of reliability, sequencing, and timeliness. However, it is clear that the nature of user traffic in coming years will become increasingly multimedia oriented and heterogeneous — for example, MPEG flows with multiple priority frames, multiplexed audio and video, multiple HTTP requests for text pages and images over the same transport connection, etc. For such *heterogeneous packet flows*, *different packets in the same flow will have different quality of service requirements in terms of priority, reliability, and timing*.

Current Internet transport protocols do not support the concept of heterogeneous flows; for example, TCP guarantees reliable and sequenced delivery for all packets in a flow, while UDP does not guarantee either delivery or sequencing for any packet. Thus, applications that have heterogeneous flows are forced to use multiple independent (homogeneous) flows and then explicitly synchronize them above the transport layer [1], [5], [6] — this is both complex for applications, and less efficient for adapting to the dynamics of the network.

Recently proposed alternatives to TCP and UDP include user-level transport protocols (e.g. RTP[7] and XTP[8]) for multimedia-oriented applications, and application level framing (ALF)[2]. RTP provides congestion control, flow control, and timing functionality in a user-level protocol on top of UDP, while ALF moves much of the transport-level functionality into

the application in order to enable application-specific handling of packet flows. We concur with the general principles enunciated by both ALF and RTP that applications should have the ability to specify the *policies* for framing, reliability, timing, and priority at the granularity of application-specific frames for their packet flows. We also believe that there should be a clean separation between *policies* and *mechanisms* — applications should specify the policies (at the application-specific frame level rather than at the packet flow level), but the transport protocol should provide the mechanisms to implement these policies. Enabling applications to specify policies at a granularity that is finer than a packet flow allows for heterogeneous packet flows; providing the mechanisms of the transport layer in the kernel rather than in the application (or a user-level protocol) allows for efficient mechanisms and simpler applications.

In this paper, we describe a transport protocol called HPF for effectively supporting *heterogeneous packet flows* in an Internet environment [3]. HPF has four key features: (a) support for reliable and unreliable packets with different priority and timing (delay) requirements in the same transport connection, (b) support for application-level framing, and for applications to specify the priority, reliability and timing requirements of each frame, (c) use of application-specified priorities as hints for network routers to preferentially drop low-priority packets during congestion, and (d) decoupling of the congestion control and reliability mechanisms in order to support congestion control for unreliable and heterogeneous packet flows. In this paper, we focus on those aspects of HPF that relate directly to the support of heterogeneous packet flows, i.e. the first three features.

The rest of the paper is organized as follows. Section II briefly discusses the goals of HPF. Section III describes the HPF architecture and design. Section IV presents a performance evaluation of HPF, and Section V concludes the paper.

II. MOTIVATION AND GOALS

Applications typically deal with data in terms of blocks or 'frames' which may have a different size than network-level packets. For heterogeneous packet flows, different frames may have different quality of service requirements in terms of reliability, priority, and deadlines. At the same time, some functions such as connection management and flow control are not frame-specific attributes. Thus, the goal of HPF is to provide mechanisms for connection establishment, flow control and congestion control on a per-flow basis, and mechanisms for reliability, sequencing, framing, prioritization, and timing requirements on a per-frame basis, where the per-frame policy is specified by the

application. HPF seeks to provide a clean separation between policies (controlled by the application), and mechanisms (controlled by the transport layer in the kernel). In summary, HPF seeks to combine the flexibility of heterogeneous packet flows with the efficiency of homogeneous packet flows.

While HPF does not require any special mechanisms within the network in order to operate, it passes down the application-defined hints about packet/frame priorities to the network. End-to-end congestion control mechanisms take on the order of a few round trips to take effect. Thus, while end-to-end congestion control can be used as a measure to adapt to longer term network variations, the network typically reacts to short-term (instantaneous) network dynamics by dropping packets during congestion. For heterogeneous packet flows, the network can improve the end-to-end perception of network quality by preferentially dropping low priority packets instead of higher priority packets [4]. Note that priority-based packet dropping will not change the number of packets dropped, but it can reduce or eliminate the number of high priority packets dropped.

In addition to supporting heterogeneity in frame/packet level requirements, HPF must also provide effective congestion control mechanisms at the transport layer. In TCP, congestion control is closely coupled with reliability—cumulative acknowledgements serve as feedback for both purposes. However, since HPF supports both unreliable and reliable packets in a single flow, congestion control is decoupled from reliability in HPF. While the details of the congestion control algorithm are beyond the scope of this paper, we refer the reader to [3].

III. HPF ARCHITECTURE AND DESIGN

HPF is a connection-oriented transport protocol. It allows applications to specify blocks of data called 'frames', and to provide frame-specific policies for reliability, priority and timing (we call these the policy parameters). A frame is then treated as a single entity, i.e. all the packets belonging to the same frame will have the same policy parameters, and a frame may be read or written as a single unit by the application. While future designs of HPF will include an option for out-of-order delivery of frames to applications, the current design of HPF guarantees in-sequence delivery (though unreliable frames/packets may be lost). HPF provides flexible policies for applications to read and write data in terms of either data streams or frames.

Figure 1 shows the overview of the HPF architecture. HPF is composed of three logical sub-layers.

- **Application framing (AF) sub-layer.** The AF sub-layer is responsible for converting frames into packets at the sender and packets into frames at the receiver, prioritization of frames and packets, and providing the application interface to read and write frames and packets with frame-specific policies.
- **Windowing, reliability, timing and flow-control (WRTF) sub-layer.** The WRTF sub-layer is responsible for coordinating the window advancement between the sender and the receiver, flow control, reliably transmitting loss sensitive packets, deleting deadline-based packets which cannot meet their deadlines, and for providing sequencing for the heterogeneous packet flow.
- **Congestion control (CC) sub-layer.** The CC sub-layer is responsible for congestion control, and for the estimation of the rate and round-trip time parameters for a connection.

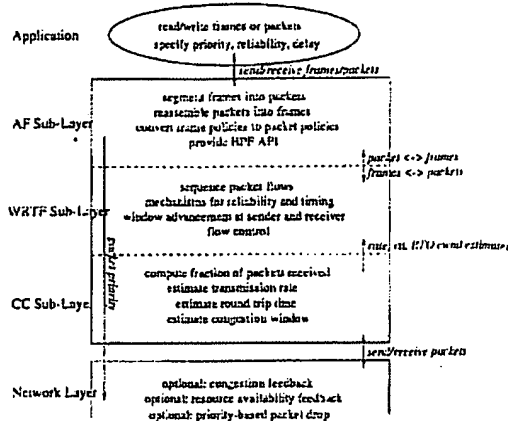


Fig. 1. HPF has three logical sub-layers.

The three sub-layers perform distinct tasks, but each sub-layer uses information from the other two sub-layers. We focus on the first two sub-layers in this paper.

A. Application Framing Sub-Layer

We describe the overview of the AF sub-layer below, and revisit the application interface in Section III-C.

At the sender, the AF sub-layer provides the applications with the ability to write frames with different policies for reliability, priority, and timing. Briefly, the AF layer takes a frame written by the application and breaks it up into packets of size MSS bytes (where MSS, the maximum segment size, is negotiated during connection establishment as in TCP). Each packet has a sequence number (which is the byte sequence number of the start of the packet, similar to TCP), and the following parameters:

- **1 bit reliability field:** if set, the packet is reliable, otherwise the packet is unreliable.
 - **n bit priority field:** 0 to $2^n - 1$ in descending order. Since HPF guarantees sequencing, reliable packets have a priority level of 0 (highest). (Our implementation currently uses $n = 2$).
 - **16 bit delay field:** the delay in milliseconds that the packet can tolerate. A delay of 0 indicates a packet with no delay bound. There are three types of packets in HPF: (a) *reliable* packets, which do not have a delay bound but have the reliability field set, (b) *unreliable delay-bounded* packets, which have a non-zero delay bound, and (c) *unreliable best-effort* packets, which have the delay field set to 0, and have the reliability field reset.
 - **1 bit frame field:** if set, indicates the end of a frame. HPF allows applications to read and write data as frames or streams. In stream mode, each packet has the frame field set. In frame mode, only the last packet of each frame has this field set.
- All packets of a frame contain the same policy parameters, which are specified by the application policy for the frame. The AF sub-layer sets the frame bit of the last packet of a frame, and adds an optional field to the header of this packet containing the sequence number of the start packet of the frame. Thus, upon reception of the last packet of a frame, the receiver can begin

reconstructing the frame. Once the AF sub-layer creates a sequence of packets for a frame, it passes this packet sequence to the WRTF sub-layer. Note that there is no copying of data involved in passing packets from one sub-layer to another, since all the sub-layers use the same send/receive buffer for the connection.

B. Window, Reliability, Timing, and Flow Control Sub-Layer

The WRTF sub-layer is the core of HPF because it provides the crucial functions of managing packets with heterogeneous requirements, and flow control. Specifically, the WRTF sub-layer performs the following tasks.

- **Sequencing:** The WRTF sub-layer guarantees that packets are delivered in sequence to the application.
- **Deadline tagging:** The WRTF sub-layer converts the relative deadline for a packet (from the AF layer) to an absolute deadline, and associates this deadline with the packet.
- **Retransmissions:** The WRTF sub-layer handles retransmissions for lost packets depending on the packet type. A reliable packet is retransmitted until it has been acknowledged. An unreliable delay-bounded packet is retransmitted until it has either been acknowledged or its deadline has expired. An unreliable best-effort packet is never retransmitted.
- **Flow control:** As in TCP, the WRTF sub-layer uses receiver-initiated 'cumulative' end-to-end acknowledgements in order to advance its sender window. However, unlike TCP, HPF has interleaved reliable, unreliable delay-bounded, and unreliable best-effort packets. Thus, the semantics of cumulative acknowledgements are different in HPF. We describe the HPF flow control algorithm in more detail below.
- **Connection establishment and teardown:** Connection establishment and teardown in HPF are identical to TCP. The SYN, SYN+ACK, ACK, FIN, and FIN+ACK control packets are marked high priority.

There are two key aspects of the WRTF sub-layer: *retransmissions*, and *flow control*. We describe each below.

B.1 Retransmissions

The retransmission policy for reliable and unreliable best-effort packets is simple: a reliable packet is retransmitted till it is acknowledged, while an unreliable best effort packet is never retransmitted.

The retransmission policy for an unreliable delay-bounded packet is more complex. While the WRTF sub-layer estimates that the packet can be retransmitted and acknowledged before its deadline, the packet is retained in the retransmission queue. At any time, WRTF sub-layer has the current estimates for the transmission rate, round-trip time, and retransmit timeout values from the CC sub-layer. At some time t , consider an unreliable delay-bounded packet p of size b bytes with deadline d , transmission rate of r bytes/ms, and round trip time of T ms. Thus, the slack, s , for the packet p is $s = d - t$ in ms. The WRTF sub-layer will keep p in its retransmit queue if $(b/r + T < s)$, i.e. WRTF estimates that p can be retransmitted and acknowledged before its deadline. Once WRTF determines that p cannot meet its deadline, it discards p from its retransmit queue.

Note that since r and T are estimates, HPF does not provide delay guarantees for packets. Of course, HPF cannot enforce

delay bound since it is purely an end-to-end protocol. However, the CC-sublayer does provide fairly accurate estimates of r and T when the dynamics of the network are not severe.

B.2 Flow Control and Window Advancement

Since HPF guarantees in-sequence delivery of packets, the window advancement algorithm at the sender and receiver are closely related to the retransmission policy described above.

Figure 2 shows the flow control and window advancement parameters in HPF. At the receiver side, the packets between $read_next$ and rcv_next can be passed up to the AF sub-layer (possibly with holes due to lost unreliable packets). The rcv_next packet has not been received, and is either a reliable packet or an unreliable delay-bounded packet whose deadline has not expired. At the sender side, all packets up to snd_una have been acknowledged and may be discarded. The key issue in flow control and window advancement is to determine how rcv_next and snd_una are advanced at the receiver and sender respectively.

For simplicity of presentation, we consider packets to be sequenced in increments of 1 rather than following byte sequencing. We describe the window advancement algorithm for three cases, in increasing degree of complexity.

Case 1: All packets are reliable: In this case, window advancement in HPF is identical to TCP. When a receiver receives a packet p_s with sequence number s , it can set rcv_next to $s + 1$ if the current value of rcv_next is s . Likewise, if the current value of rcv_next is s and the receiver has already received the packet p_s , then the receiver can advance rcv_next to $s + 1$.

Case 2: Packets are either reliable or unreliable best-effort: For this case, the semantics of rcv_next are the following: every reliable packet with a sequence number less than rcv_next has been received. Consider a packet $p_{s,h}$, where s denotes the sequence number of the packet and h denotes the sequence number of the last reliable packet preceding $p_{s,h}$. The receiver can set rcv_next to $s + 1$ if the current value of rcv_next is greater than h . We say that $p_{s,h}$ depends on the packet with sequence number h . Likewise, if the current value of rcv_next is h , then the receiver can update the value of rcv_next to the highest sequence number s , such that (a) the packet $p_{s,h}$ has been received, and (b) the transitive closure of the packets on which $p_{s,h}$ 'depends' has been received.

In essence, the receiver can advance its receiver window beyond the last packet p it has received such that every high priority packet preceding p has been received. The loss of a reliable packet will stall the progress of the receiver window. On the other hand, the loss of an unreliable best-effort packet will not stall the progress of the receiver window, though it will cause a hole in the sequence of packets sent to the AF sub-layer at the receiver. Delayed out-of-sequence unreliable best-effort packets are discarded, because the receiver window would have already advanced beyond the sequence numbers of such packets when they are received.

Case 3: Packets are reliable, unreliable delay-bound, or unreliable best-effort: This case is more complicated than the ones

before because delay-bound packets may be retransmitted, but cannot be treated like reliable packets since they may be discarded upon violation of their delay bounds. Thus, the loss of an unreliable delay-bound packet may or may not require the receiver window to advance, depending on whether the deadline of the packet has expired or not. However, the receiver has no notion of when the deadline of a delay-bound packet has expired. Thus, the sender must treat the unreliable delay-bound packet like a reliable packet while its deadline has not expired and like an unreliable best-effort packet after its deadline has expired, and provides notifications to the receiver on how far it can advance its receiver window.

Consider a packet $p_{s,h,w}$, where s denotes the sequence number of the packet, h denotes the sequence number of the last reliable packet preceding $p_{s,h,w}$, and w denotes a sequence number specified by the sender. When the receiver receives the packet $p_{s,h,w}$, it will set rcv_nxt to $\max(rcv_nxt, w)$ if the current value of rcv_nxt is greater than h . Likewise, if the current value of rcv_nxt is h , then the receiver can update the value of rcv_nxt to the highest sequence number w , such that (a) the packet $p_{s,h,w}$ has been received, and (b) the transitive closure of the packets on which $p_{s,h,w}$ "depends" has been received.

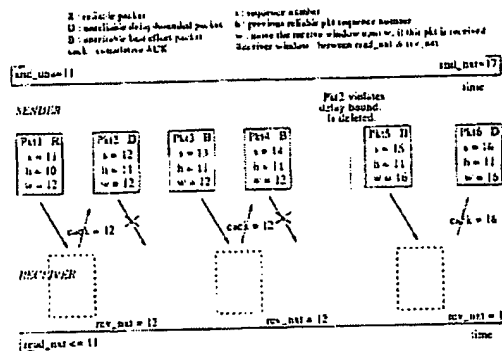


Fig. 2. Illustration of the window advancement algorithm in HPF.

In essence, the sender uses the value w in each packet to control window advancement at the receiver. For an unreliable delay-bound packet p with sequence number s , while it has not been discarded from the retransmit queue, the w field of each outgoing packet will be upper bounded by s . Once p is discarded from the retransmit queue due to delay violation (or if it has been acknowledged), the w value in subsequent packets transmitted from the sender can be greater than s . The receiver sets its receiver window to the maximum w among all those received packets p , such that the transitive closure of packets on which p depends has been received. Figure 2 illustrates a simple example of this algorithm.

Note that Case 1 is a special case of Case 3 when a packet with the sequence number of s can be denoted by $p_{s,s-1,s+1}$, and Case 2 is a special case of Case 3 when a packet with the sequence number of s that depends on a packet with the sequence number of h can be denoted by $p_{s,h,s+1}$. In the WRTF sub-layer, we implement the approach described in Case 3 for window advancement at the receiver, and the re-

transmit algorithm in Section III-B.1 at the sender.

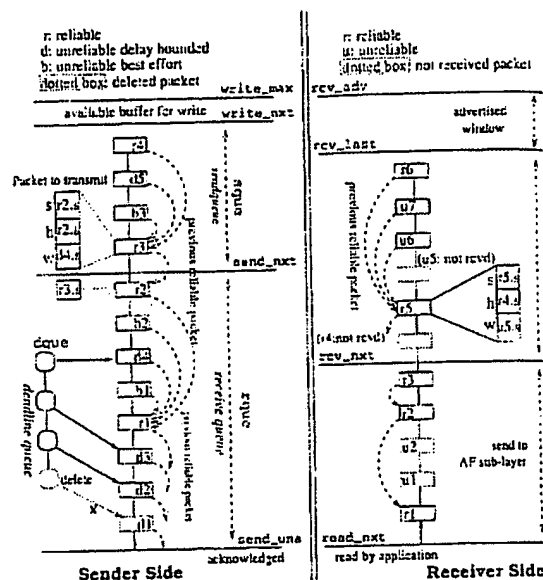


Fig. 3. Illustration of a snapshot of the WRTF state with respect to window advancement and flow control.

Figure 3 shows a typical snapshot of the state at the WRTF layer. Each packet $p_{s,h,w}$ contains three sequence numbers: $p.s$, $p.h$, and $p.w$. Determining $p.h$ for a packet is simple – the sender maintains the sequence number h of the last high reliable packet that has been queued for transmission, and every subsequent packet that is queued for transmission has $p.h = h$. The sender maintains three queues of packets: a send queue $sque$ which queues the packets written to by the sender but not yet transmitted, a retransmit queue $rque$ which queues packets that can be retransmitted, and a deadline queue $dque$ which is a queue of pointers to the unreliable delay-bounded packets in the retransmit queue. All queues are ordered by sequence number.

The $p.w$ value of an outgoing packet is the sequence number of the head of line packet in $dque$. The retransmit policy governs when a delay bounded packet is deleted from the send/retransmit queue.

Once the receiver computes the rcv_nxt value, it can compute the advertised receiver window size from Figure 3. A WRTF-acknowledgement from the receiver consists of two parts: the cumulative acknowledgement ($cack$), and the advertised window (adv). The receiver sets $cack$ to rcv_nxt , and adv to $rcv_adv - rcv_last$ in its WRTF-acknowledgements. When the sender receives a WRTF-acknowledgement from the receiver, it sets snd_una to $cack$, and sets $write_max$ to $snd_nxt + \max(adv, cwnd)$. The congestion window, $cwnd$, is computed by the CC sub-layer and passed on to the WRTF sub-layer.

Figure 4 provides an outline of the pseudo-code for the window management and flow control algorithms in HPF.

Support for delay-bounded packets in HPF does not require the receiver to know which packets are delay bounded, or main-

```

transmit(p, count) /* transmit packet p; p has been transmitted count times before */
dequeue(p, all) /* dequeue packet p from all the queues to which it belongs */
increment count for p
update(dqmc) /* iteratively purge HOL delay-bounded packets that have violated
              (or will violate their deadline */
if (count == 1) /* first time transmission */
    p.h <- h /* set last reliable sequence number */
    if reliable(p)
        h <- p.s
if not_empty(dqmc)
    p.w <- dqmc.HOL -> s
else
    /* w is an upper bound on the window increase at the receiver */
    p.w <- p.s + 1
if reliable(p)
    enqueue(p, rque)
else if delay-bounded(p) and can_retransmit(p)
    enqueue(p, rque)
    enqueue(p, dqmc)
else discard(p)
update(dqmc) /* update the deadline queue */
while not_empty(dqmc) and delay_violation(dqmc.HOL)
    delete(dqmc.HOL) /* causes dqmc.HOL to be deleted from dqmc and rque */
receive(p) /* receive function at the receiver */
if reliable(p) and (p.h == r) /* r is the last reliable packet received such */
    /* that all reliable packets < r have also been received */
    r <- p.s
    ptr <- enqueue(p, revque) /* enqueues p in sequence; returns pointer to p in revque */
    while(ptr != NULL) /* chain through received packets */
        if reliable(ptr) and (r >= ptr->h)
            r <- ptr->s /* maintains invariant of r */
else
    enqueue(p, revque) /* for unreliable packets */
ptr = revque.HOL;
while (ptr != NULL) and (ptr->h <= r) /* if all reliable packets preceding ptr */
    /* have been received, then set rev_next */
    rev_next = next(rev_next, ptr->w) /* to max of rev_next and ptr->w */
    /* to max of rev_next and ptr->w */

```

Fig. 4. Pseudo-Code for key pieces of the Sender and Receiver Flow Control Algorithm.

tain any kind of clock at the receiver. As a result of our minimalist approach, the timing semantics for delay-bounded packets in HPF are 'mostly within deadline' delivery rather than 'guaranteed within deadline' delivery for those packets that are received at the receiver. This is an acceptable model for most applications that need real-time support in the Internet. Thus, HPF can support heterogeneous flows with interleaved reliable, unreliable delay-bounded, and unreliable best-effort packets, in which the applications specify the policies but the mechanisms are provided in the kernel. Figure 2 shows an example of how this approach works.

C. Application Interface

We now revisit the AF sub-layer for the purposes of describing the application interface for HPF. As mentioned before, a unique feature of HPF is that it allows applications to specify per-frame policies for reliability, priority, and timing. HPF provides a socket interface very similar to the stream socket interface for TCP. Each application creates and binds sockets with the `socket()` and `bind()` calls respectively. For HPF, we specify a new socket type called `SOCK_HPF`. The server then makes the `listen()` and `accept()` calls, while the client makes the `connect()` call. Once an HPF connection is established, HPF provides special mechanisms for application-level framing, and the specification of per-frame reliability, priority, and timing requirements, as described below.

An application may read from or write to a HPF connection in two modes: *stream mode*, or *frame mode*. In stream mode, data is transmitted over the network as a continuous stream of

bytes (with holes in between where unreliable packets are lost). In frame mode, data is transmitted over the network as a sequence of frames (with holes in between where parts of unreliable frames are lost – the loss of any part of a frame causes the loss of the entire frame). In the same flow, an application may interleave the two different modes when reading or writing. Besides, the writer and reader need not be in the same mode: in particular, the writer may be in frame mode while the reader may be in stream mode. Figure 5 shows the difference between reading and writing in the two modes.

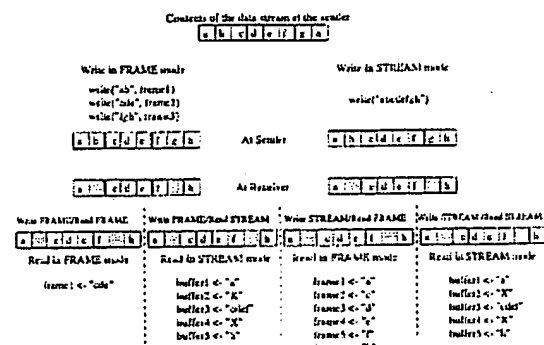


Fig. 5. Read/Write in different modes. This figure shows the different responses that applications will see depending on the read and write modes. The shaded box indicates a lost packet. An 'X' in a receive buffer indicates invalid data.

Writing to a HPF connection: An application writes to a HPF connection using the `HPFwrite()` socket call, similar to the `write()` socket call, but with additional fields. The `HPFwrite()` call has the following policy parameters:

- **socket descriptor, buffer, length:** These are the standard write parameters.
- **mode:** The mode field can be either `STREAM` or `FRAME`. In stream mode, each packet is considered to be an independent unit, while in frame mode, all the packets belonging to the same frame are considered to be a single unit.
- **reliability field:** The reliability field may be either `RELIABLE` or `UNRELIABLE`.
- **priority field:** The priority field can be 0, 1, 2, or 3, in descending order of priority.
- **frame field:** In stream mode, the frame field is irrelevant since the AF sub-layer will set the frame field to 1 for each packet. In frame mode, the application will set the frame field to 1 to denote the end of a frame, and 0 otherwise. Thus, multiple writes may constitute a single frame.
- **delay field:** The delay field is a 16 bit unsigned short integer denoting the delay bound (relative to the time of writing) in milliseconds.

In frame mode, the AF sub-layer will buffer the entire frame before packetizing it and forwarding it to the WRTF sub-layer. For a frame that is written using multiple `HPFwrite()` calls, only the frame policy specified in the last write are significant, i.e. all packets in the frame will contain the same options specified in the last write for the frame.

Reading from a HPF connection: An application reads from a HPF connection using the `HPFread()` socket call, similar to the `read()` socket call, but with five fields: socket descriptor, buffer, length, mode, and status. The status field is a pointer to an unsigned character, and is a return parameter.

If the mode field is `FRAME`, then the `HPFread()` call fills the buffer with the next completely received frame, and returns the size of the buffer read. If the buffer size is less than the frame size, the remaining contents of the frame are lost, and an error notification is set through the status field. Note that any partially received frames are discarded at the receiver.

If the mode field is `STREAM`, then the `HPFread()` call returns a maximal sequence of either valid data or invalid data, bounded in size by the 'length' parameter of the `HPFread()` call. By maximal sequence, we mean that either all the data returned by a read call is valid, or all of it is invalid (in case of lost packets in the packet flow). The return value of the `HPFread()` call specifies the size of the read, and the status field indicates whether the buffer contains valid data or invalid data.

If the writer is in stream mode, then each individual packet is treated like a separate frame and it is immaterial whether the reader is in stream or frame mode. However, if the writer is in frame mode, the reader will experience different responses from the connection depending on whether it is in stream mode or frame mode. If the reader is in stream mode, it can read parts of a frame even if some other parts of the frame were lost. However if the reader is in frame mode, it can read a frame only if the entire frame were correctly received. Note that the reader can migrate between the stream mode and the frame mode. Figure 5 shows an example of different read and write modes.

Retrieving Connection State: In order to support application adaptation, we provide the ability for the application to retrieve the current rate and expected latency of the connection via `getsockopt()` calls. The latency value returned is half the estimated RTT, while the rate value returned is $\max\{ssthresh, cwnd/RTT\}$. These parameters are designed to serve as coarse estimates for the application to adapt in the long term.

D. TCP backward compatibility

HPF uses a 3-way handshake protocol for connection establishment identical to TCP. In order to preserve backward compatibility with TCP for homogeneous reliable flows, the client and the server negotiate the protocol by means of the following mechanism during connection establishment: HPF includes special options after the TCP header in the SYN and SYN+ACK packets. An application can enable HPF options via a `setsockopt()` call prior to the `connect()` or `accept()` calls. If both end-points include the HPF option fields during connection setup, then an HPF connection is established. Otherwise a standard TCP connection is established.

IV. PERFORMANCE MEASUREMENTS WITH HPF

We present a preliminary performance evaluation of HPF. In particular, we are interested in comparing HPF with TCP and

UDP under congestion. Performance comparison to RTP is ongoing work.

In this section, we present four tests. First, we compare TCP with HPF with random packet drops in the network ranging from 5% to 20%. Second, we compare the goodput of TCP and HPF for different ratios of high and low priority packets in a heterogeneous packet flow. Third, we run tests over the Internet to measure impact of HPF over long-haul connections. In the absence of network support for priority-based packet dropping, priorities for unreliable packets have no impact on the performance of HPF. Test 1 - 3 were performed without assuming any special network support, and thus we only used two levels of priority - high for reliable packets and low for unreliable packets. For test 4, we instrumented the router to support priority-based packet dropping in order to demonstrate the coordinated adaptation of HPF and the network for an MPEG flow with multi-priority frames. We have not considered unreliable delay-bounded packets in the performance evaluation section of this paper.

The testbed environment is shown in Figure 6. The dedicated network consists of star topology with four computers all running Linux 2.0.33, and connected via point-to-point 10 Mbps Ethernet links. Note that, we have throttled the link between hosts *airi* and *durga* to approximately 1 Mbps by introducing a 10ms delay between successive packet transmissions in order to allow us to create network congestion by overloading this bottleneck link with packets.

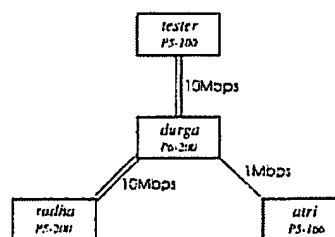


Fig. 6. The experimental testbed configuration used for the performance tests. The link between hosts *durga* and *airi* is throttled in order to create congestion, reduce the apparent bandwidth of the link, and increase the delay for packets traversing through the link.

A. HPF versus TCP with different random packet drop percentages

In the first test, we compare the performance of HPF and TCP with different percentages of random packet dropping. Table 1 shows the total time in seconds spent to receive 2MB of data for both TCP and HPF. Data traffic goes from *radha* to *airi* without congestion in this test. 12.5% of packets are of high priority. We modify the kernel on *radha* such that it randomly selects 0%, 5%, 10% and 20% of packets to drop in four different tests. From the 0% loss experiment, we notice that HPF introduces an overhead of 0.004 seconds for a 2 MB file transfer. As the percentage of packets dropped starts to rise, HPF performs faster than TCP. Since dropping a low priority (unreliable) packet does not trigger a retransmit, the total time needed to complete the file transfer with 5% loss is smaller than in the no loss case. In fact, as long as the dropping probability is less than the percentage

of high priority packets, the cost of retransmitting high priority packets is compensated by the smaller number of low priority packets received. Note that our comparisons of HPF and TCP in this experiment are inherently unfair with packet drops because HPF is transferring fewer bytes than TCP, which is reliable. The purpose of this example is not to show that HPF is faster than TCP, but that the rate of data transfer of the high priority (reliable) packets degrades more gracefully for HPF than TCP under packet loss. Thus, for applications that can tolerate loss, HPF degrades more gracefully with increase in packet loss.

TABLE I
THE SPEEDUP OF HPF VERSUS TCP WITH DIFFERENT PERCENTAGES OF RANDOM DROP.

Percent	TCP (Sec)	HPF (Sec)	HPF bytes dropped	TCP/HPF speedup
0%	13.806	13.810	0%	≈ 1
5%	24.56	13.37	4.5%	1.84
10%	53.26	13.75	8.4%	3.88
20%	149.91	17.33	16.3%	5.65

B. HPF versus TCP with different priority ratios

For the second test, three different 2MB data flows were transmitted from *radha* to *atri* using various ratios of high priority (reliable) to low priority (unreliable) packets. During the transmission, three unregulated flows offered congestion in the form of multiple bursts of UDP packets at 60 millisecond intervals from *tester* to *atri*. The length of time required to complete the transmission of the 2MB data flows was then recorded over multiple runs. Table II shows the results, averaged over the three data flows. The different combinations of the High:Low ratio used applies to all of the data flows. Note that as the fraction of low priority packets increases, HPF performs faster than TCP at the expense of low priority packet loss. Specifically, for the High:Low ratio of 5:95, we observed a 41% reduction in the overall transmission time of the data flows at the expense of a 23% loss in low priority packets and no loss in high priority packets (which are retransmitted till successfully acknowledged).

As in the previous example, our intention is not to perform a direct head-to-head comparison of HPF against TCP because the two protocols are not transmitting the same amount of data. The goal is to show how applications perceive the progress of flows with different High:Low packet ratios in scenarios with high burst losses.

TABLE II
THE PERFORMANCE OF HPF VERSUS TCP FOR VARIOUS HIGH:LOW PRIORITY RATIOS WITH MULTIPLE CONCURRENT STREAMS.

Protocol	High:Low Ratio	Packets Dropped	Time improvement vs TCP
TCP	all high	0%	
HPF	50:50	4.9710%	9.9421%
	33:66	7.6238%	16.8454%
	20:80	12.5544%	17.7903%
	10:90	16.9267%	34.7568%
	5:95	23.7603%	41.5078%

C. HPF versus TCP over the Internet

For the third test, we created a one way tunnel from UIUC to Boston with 15 hops. We send HPF flows with different High:Low ratios ranging from 100% high priority packets (TCP-like) to 100% low priority packets (UDP-like). Table III shows the results. While UDP (not shown in table) performs very badly in this case, losing as many as 75% of the packets due to unregulated transmission, HPF provides a very attractive trade-off between the number of packets lost and transmission rate. In particular, even with 100% low priority packets, HPF lost only about 8% of the packets but improved the apparent throughput of the connection by about 75% over TCP. Of course, HPF performs significantly better than UDP since it performs congestion control. The congestion control algorithm of HPF is also the reason for its superior performance over TCP even for the scenarios with small fractions of packet loss. This test shows that even over the Internet, without any link-layer support for priority dropping, HPF can significantly improve the apparent quality of a connection for loss tolerant flows.

TABLE III
THE PERFORMANCE OF HPF VERSUS TCP FOR VARIOUS HIGH:LOW PRIORITY RATIOS OVER AN IP TUNNEL.

Protocol	High:Low Ratio	Packets Dropped	Improvement vs TCP(time)
TCP	all high	0%	—
HPF	66:33	0.71%	31.77%
	50:50	2.86%	51.60%
	33:66	4.29%	64.76%
	10:90	5.71%	69.08%
	5:95	7.86%	69.93%
	0:100	7.86%	75.07%

D. Impact of priority-based packet dropping in the network on HPF

For the fourth test, we added network support in the form of a 'HPF priority-aware' Round Robin (RR) packet scheduler that provides a separate FIFO queue for each flow, and priority-based packet dropping upon buffer overflow. For this test, we ran the modified scheduler at *durga*, and we created two HPF flows from *radha* to *atri* and two from *tester* to *atri*. The High:Low ratio was set to 5:5 for all flows. The result was that the time improvement over TCP for this case was 57.9% as opposed to 29.7% without the modified scheduler. Note that the number of packets dropped at *durga* did not change; however, lower priority packets were dropped rather than higher priority packets, thus improving both the perception of the connection quality and the throughput (due to fewer retransmissions).

As a further illustration of the use of priority-based packet dropping in the network with HPF, we ran a modified VCR client-server application that sends MPEG-1 streams over HPF [1] with and without priority-based packet dropping. The modified VCR program uses reliable packets for control information and unreliable packets with descending levels of priority for I-frames, P-frames, and B-frames respectively.

We ran two experiments with a VCR client at *atri* and a VCR server at *radha*, and introduced periodic UDP packet bursts from *tester* to *atri* in order to cause congestion over the *durga* to *atri*

link. In the first experiment, *durga* ran a Round Robin packet scheduler but without priority-based packet dropping. In this case, we observed that the loss in the I-frames was 20%, P-frames was 20%, and B-frames was 8%. This result is intuitively justifiable, since I-frames and P-frames are larger than B-frames, and the loss of even one packet in a frame led to the loss of the entire frame (the client reads in frame mode). In the second experiment, *durga* ran a Round Robin packet scheduler with priority-based packet dropping. In this case, we observed that the loss in the I-frames was 0%, P-frames was 0%, and B-frames was 24%. This result was induced by the fact that lower priority packets belonging to B-frames were preferentially dropped during congestion. This simple test points to the obvious usefulness of augmenting HPF at the end hosts with network support in the form of priority-based packet dropping.

V. CONCLUSION

In this paper, we have described three key features of HPF:

- The ability of HPF to support heterogeneous packet flows, wherein different packets/frames may have different quality of service requirements in terms of reliability, priority, and deadlines.
- The ability of HPF to support application-level framing, and provide APIs for applications to specify the priority, reliability and timing requirements of each frame.
- The ability of HPF to propagate application-specified priorities as hints for network routers to preferentially drop low-priority packets during congestion.

We have implemented and used HPF for close to a year now. From our experiences with HPF, we have observed the following:

- HPF works fairly well without any network feedback, and substantially improves the apparent network quality for loss tolerant flows. HPF improves significantly with even minimal network support. In particular, priority-based packet dropping in the network augments the performance of HPF perceptibly.
- HPF can support delay-bounded packets without requiring any clocks at the receiver. This is because of the 'mostly before deadline' semantics of delay-bounded packet delivery in HPF, and also because deadlines for HPF packets include the time for the acknowledgement from the receiver to reach the sender. This is a very attractive feature for supporting real-time flows in an Internet environment.
- HPF can eliminate the problems associated with synchronization of multiple streams of packets because it can support heterogeneous packet flows, and it provides sequencing of such flows.
- We have found that in terms of performance, applications which specify per-frame policy parameters to HPF and let HPF do the adaptation consistently performed better than applications that performed adaptation themselves - either running over HPF or running over UDP. This is a direct consequence of the fact that the adaptation mechanisms of HPF are kernel-level, and hence more efficient.

REFERENCES

- [1] S. Cen, C. Pu, R. Stachli, C. Cowan, and J. Wolpole. A distributed real-time mpeg video audio player. In *Fifth International Workshop on Network and Operating System Support of Digital Audio and Video (NOSSDAV)*, April 1995.
- [2] D. Clark and D. Tennenhouse. Architectural considerations for a new generation of protocols. In *Proceedings of ACM SIGCOMM*, pages 200-206, August 1990.
- [3] D. Dwyer, J.-R. Li, S. Ha, and V. Bharghavan. An adaptive transport protocol for multimedia communication. In *IEEE International Conference on Multimedia Computing Systems*, June 1998.
- [4] S. Ha, K.W. Lee, and V. Bharghavan. Performance evaluation of scheduling and resource reservation algorithms in an integrated packet services network environment. In *IEEE Symposium on Computers and Communications*, July 1998.
- [5] S. McCanne and V. Jacobson. vic: A flexible framework framework for packet video. In *ACM Multimedia*, November 1995.
- [6] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven layered multicast. In *Proceedings of ACM SIGCOMM*, August 1996.
- [7] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A transport protocol for real-time applications. RFC: <http://ds.internic.net/rfc/1889.txt>, January 1996.
- [8] A.C. Weaver. Xpress transport protocol specification v.4.0. *XTP Forum*, March 1995.



INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Experiments with a Layered Transmission Scheme over the Internet

Thierry Turletti Sacha Fosse Parisis Jean-Chrysostome Bolot

N° 3296

Novembre 1997

THÈME 1





Experiments with a Layered Transmission Scheme over the Internet

Thierry Turletti Sacha Fosse Parisis Jean-Chrysostome Bolot

Thème 1 — Réseaux et systèmes
Projets Rodeo

Rapport de recherche n° 3296 — Novembre 1997 — 26 pages

Abstract: Combining hierarchical coding of data with receiver-driven control appears to be an attractive scheme for the multicast transmission of audio/video flows in a heterogeneous multicast environment such as the Internet. However, little experimental data is available regarding the actual performance of such schemes over the Internet. Previous work such as that on receiver driven layered multicast uses join experiments to choose the best quality signal a receiver can subscribe to. In this paper, we present a receiver-based multicast rate control mechanism based on a recently proposed TCP-friendly unicast mechanism. We have implemented this mechanism and evaluate its performance in conjunction with a simple layered audio coding scheme. We find that it has interesting convergence and performance properties, but also bring out its limitations.

Key-words: Audio coding, congestion control, hierarchical coding, Internet, multicast transmission.

Unité de recherche INRIA Sophia Antipolis

2004, route des Lucioles, B.P. 93, 06902 Sophia Antipolis Cedex (France)

Téléphone : 04 93 65 77 77 - International : +33 4 93 65 77 77 — Fax : 04 93 65 77 65 - International : +33 4 93 65 77 65
à partir du 01/01/1998

Téléphone : 04 92 38 77 77 - International : +33 4 92 38 77 77 — Fax : 04 92 38 77 65 - International : +33 4 92 38 77 65

Etude et mise en oeuvre d'un schéma de transmission hiérarchique sur Internet

Résumé : La transmission audio/vidéo en multipoint dans un environnement hétérogène comme l'Internet soulève de nombreux problèmes. L'utilisation d'un schéma de transmission qui combine un codage hiérarchique et un contrôle de transmission orienté-récepteur apporte une solution élégante au problème de l'hétérogénéité des récepteurs. Cependant, peu de données expérimentales sont disponibles sur les performances effectives de tels schémas de transmission dans l'Internet. Dans l'une des approches existantes, la transmission hiérarchique orienté-récepteur, le récepteur choisit le nombre de couches auquel il s'abonne en fonction de d'abonnements test préalables. Dans cet article, nous décrivons un mécanisme de contrôle de transmission en multipoint orienté-récepteur basé sur un mécanisme *TCP-friendly* récemment proposé pour la transmission en point à point. Nous avons mis en oeuvre ce mécanisme en conjonction avec un schéma de transmission audio hiérarchique et nous évaluons ici ses performances et limitations.

Mots-clés : Audio, codage hiérarchique, contrôle de congestion, Internet, transmission multipoint hiérarchique

1 Introduction

The transmission of real-time audio and video over the Internet has been much in the news recently. In particular, the transmission of voice has achieved high visibility because of technical, financial, and regulatory issues related to so-called "Internet telephones" (e.g. refer to [17, 38]), to the rapidly increasing number of companies selling or giving away Internet telephony software, and to the increasing traffic generated by these telephones. Even though most of the commercial companies involved in the Internet telephony business date the beginning of Internet telephony to the first release of VocalTec's "Internet Phone" [37] in 1995, the transmission of unicast audio dates back to the 70's [39]*, while the transmission of multicast audio started "officially" at the March 1992 IETF [4]. However, it is true that only recently has audio and video traffic made up a non-negligible fraction of the traffic routed at some nodes in the Internet. In any case, this traffic is expected to increase for at least two reasons. First, a rapidly growing number of tools are available (some of them for free) that provide low bandwidth but decent to excellent quality audio and video coding [9]. Second, such tools are being incorporated in a growing number of applications such as collaborative working applications [9].

The wide use of unicast and multicast multimedia tools in the Internet raises two important related questions. First, what is the impact of a much increased multimedia traffic on other traffic? Second, what kind of multimedia quality can users expect given that the network does not provide guarantees on delay, jitter, bandwidth, or loss rate? The impact of multimedia traffic on network and application performance is potentially quite large because the vast majority of currently available tools send data at a rate which does not depend on network state. For audio tools, this rate would be a constant rate corresponding to the chosen coding scheme (64 kb/s for G.711 coding) in the absence of silence detection. Such a behavior is unlike that of rate controlled applications, such as TCP-based applications, which adjust their output rate and hence their bandwidth requirements depending on the state of the network. Thus, non rate controlled applications are "bad citizens" since they share network resources with TCP-based applications in unfair ways, and their rapidly increasing use in the Internet might cause long-lived, severe congestion problems (especially for TCP users).

There are two ways to prevent this from happening. One way is to replace the FIFO scheduling algorithms in Internet routers with RED-like algorithms that "punish" aggressive flows and reward rate controlled applications [13]. Another way is to incorporate in applications TCP-friendly rate control mechanisms which make sure resources are not shared unfairly with TCP connections, and suitable for both unicast and multicast transmission. This latter way can be thought of as a special case (namely rate adaptation) of a more general approach which aims at adapting application behavior to network characteristics, the goal being to maximize the quality of the data delivered to the destinations. Other

* Also see the discussion on the AVT mailing started on Sept. 16, 1996 by Ed Ellenson regarding a "Packetized Audio Patent".

kinds of adaptation include adaptation to delay jitter by playout adjustment schemes [29] or adaptation to loss by ARQ- or FEC-based error control schemes [11, 33, 28].

The second way above (namely incorporating rate control mechanisms in applications) does not require modifications to router software and hence can be implemented in the current Internet. However, none of the commercial tools we are aware of uses any kind of rate control[†]. This is not so surprising after all since non rate controlled applications tend to “steal” bandwidth from rate controlled applications, and thus provide better quality to their users (this will be illustrated in Section 4). This makes it all the more important that all applications be rate controlled since the current Internet does not provide incentives to behave in a fair way.

Unfortunately, the design of rate control schemes for multicast real time applications is not a simple extrapolation of the TCP-like source based control schemes used by unicast data applications for two main reasons. First, source based rate control schemes using feedback information about the state of the network do not scale well with the number of participants in the multicast group, implying that the rate of exchange of any information between participants needs to be carefully controlled [32, 3]. Second, source based schemes do not scale well with the level of heterogeneity in the branches of the multicast tree and/or in the participants. This is because the source can adjust the rate of a stream to match the requirements of one participant (i.e. to adapt to the state of one branch in the tree), but it cannot match the conflicting requirements of multiple heterogeneous participants (i.e. adapt at the same time to different states of different branches). Various approaches to this problem have been described in the literature, using gateways [27, 1], simulcast transmission [7], and layered transmission coupled with layered coding [22, 23].

We focus on the layered transmission / layered coding approach in this paper. While this approach is attractive and has been advocated for some time now, it has not yet been deployed over the Internet (although this is expected to change real soon now). The approach relies on the availability of two components, a layered coder and a layered transmission scheme. With *layered coding*, source data is encoded into a number of layers, or (sub)bands, that can be combined to reconstruct a signal that gets closer to the original signal as the number of combined layers increases. Layered coders for audio and video have been developed over the past few years, but few coders have low enough CPU requirements as to be useful in software only tools. Recent work has produced such low CPU layered video coders [23, 5]. However, we are aware of little equivalent work for audio coders. Thus, we have developed simple layered schemes for audio based on the idea of temporal subsampling, which are described in Section 2.

The idea of *layered transmission* then is to send each layer on a separate multicast group, with receivers deciding to join/quit a group (i.e. to receive/drop another layer) on their own. The basic principles of layered transmission have been known for a while (see references in [22]). However, a specific scheme specifying how/when receivers are to join and quit layers has only recently been described and simulated. This scheme, referred to as

[†] It is hard to blame the commercial tools for behaving so as they only mimic research tools in this respect. None of the more popular Mbone tools such as VAT[36] uses any kind of rate control either...

Receiver driven Layered Multicast (RLM), uses “probing” experiments similar to those used by TCP, to decide when to join and quit layers. We propose a layered transmission scheme in which RLM’s join experiments are replaced by an explicit estimation at each receiver of the bandwidth that would be used by an equivalent TCP connection (the notion of equivalent connection will be defined later) between the source and the receiver. This estimation is done using the same technique as that recently described to design TCP-friendly unicast rate control scheme [20]. Thus, we obtain a TCP-friendly receiver-based multicast rate control mechanism. We have implemented this scheme and evaluated its performance and limitations on the MBone.

The rest of the paper is organized as follows. In Section 2, we describe simple layered audio coding schemes. In Section 3, we describe the receiver-based control scheme. In Section 4, we describe the experimental settings and discuss the results. Section 5 concludes the paper.

2 Simple layered audio coding schemes

Layered coding is a family of signal representation techniques in which the source information is partitioned into a sets called layers. The layers are organized so that the lowest, or base layer, contains the minimum information for intelligibility. The other layers, called complementary layers, contain “add-on” information which improves the overall quality of the signal. Usually, layered encoding schemes are organized so that some layers (in particular the base layer) are mandatory to reconstruct a coherent signal. Using such schemes then requires either that the net be able to discriminate between packets and provide packets that carry important information with a guaranteed performance (in particular guaranteed maximal loss rate) service, or that these packets be “protected” against loss. This can be done using a variety of so-called unequal error protecting schemes, which have been the subject of much research effort recently (e.g. [16, 10]).

Our goal was not to develop or experiment such schemes, but instead to experiment with rate control schemes. Thus, we have focused on simple coders with low cpu requirements and no need for unequal error protection schemes. We describe next a simple layered encoding scheme in which all layers have the same importance, meaning that the loss of information from one layer does not impact quality more than the loss of information from another layer.

2.1 The basic subsampling scheme

The simplest balanced scheme one can think of is based on straight subsampling. The coding algorithm is based on a temporal subband decomposition. Consider for example the case of a PCM (Pulse Coded Modulation) encoded 8kHz audio signal decomposed into 3 PCM 2.7 kHz flows as shown in Figure 1. The temporal decomposition algorithm is carried out at the source done for each audio chunk (which typically includes 20ms, 40ms or 80ms of audio). At a destination, a receiver which receives all 3 flows can retrieve the original input signal. If one or two flows are missing, the destination uses the samples received to reconstruct an

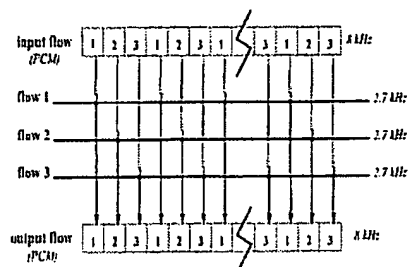


Figure 1: Hierarchical coding scheme

approximation of the original signal. An upsampling one-to-three flows is shown in Figure 2. Clearly, the larger is the number of received flows, the better the quality of the reconstructed signal.

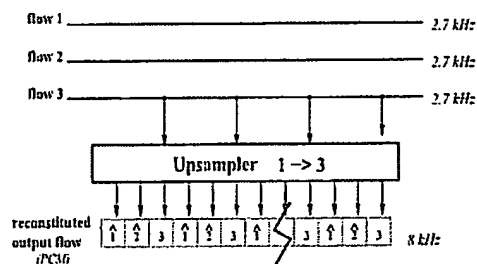


Figure 2: Reconstruction 1 to 3

Note that temporal decomposition handles signals sampled with different sampling rates. For example, a 48 kHz audio signal can be decomposed into three 16 kHz subflows, each of which can be decomposed into two 8 kHz subflows, which finally yields eighteen 2.7 kHz audio layers. Of course, temporal subband decomposition can be coupled with compression schemes[†].

2.2 Robustness against packet loss

The audio coding scheme described above has the interesting balanced property mentioned above: all the layers have the same importance in the network. However, in case of severe congestion, the congestion control algorithm may select to subscribe to only one flow and

[†] Refer to the Web page <http://www.inria.fr/rodeo/turetti/audio/> to retrieve samples showing the impact on quality of the number of received layers, and of the compression scheme used to encode the layers

in this case, the coding has no mechanism at all to reconstruct lost samples of audio. To achieve robustness against packet loss, the receiver could decide to subscribe to at least 2 flows. We have decided to add to the base flow of each sample, the flow L corresponding to the previous sample (L is the number of layers of the coding). So, in such a scheme, the first layer has twice the bandwidth of other layers and the transmission control scheme handles $(L - 1)$ flows instead of L . Note that contrary to other redundancy schemes, there is no bandwidth wasted when packet loss is null since the "redundancy" information appended to the first layer is always used to improve the reconstruction of the sample.

Note also that the redundancy added to flow # 1 does not imply that this flow has a higher priority since flow # 1 is not required to receive it correctly to decode the other flows. Whereas there is no flow more important than another one, all receivers have to adopt the same order in the subscription algorithm (in order to perform efficient pruning to limit the bandwidth sent).

2.3 CPU and bandwidth cost

The cpu cost of the hierarchical coding algorithm mainly depends of the compression scheme used following the temporal subband decomposition. In our experiments, we have successfully tested the PCM, ADM6, ADM5, ADM4, and ADM3 compression schemes, which have very low cpu requirements [14]. More efficient compression coder could as well fit into the temporal subband decomposition described above. However, very high compression coding algorithms do not appear to fit well with layered coding over the Internet because the overhead of IP, UDP and RTP[32] headers decrease significantly their bandwidth savings.

Table 1 shows the bandwidth overhead for layered and non-layered audio coding using PCM and ADM4 compression schemes. The IP/UDP/RTP bandwidth headers overhead is shown for three sizes of packets corresponding to 20ms, 40ms and 80 ms of compressed speech.

Interactivity needed by audio/video conferencing applications is often cited as requiring low packetization intervals of 20ms. However, such a low packetization interval increases the number of packets sent per second, especially if several layers are used. Then the bandwidth requirement of the IP/UDP/RTP headers may be higher than the bandwidth requirement of the actual payload, see Table 1. For example, the bandwidth corresponding to the headers for 6 flows is (for 20ms packetization interval) equals to 106 kbps with IPv4 and 144 kbps with IPv6. Table 2 recalls the size of IPv4, IPv6, UDP and RTP headers.

However, experience with audioconferencing tools in the Mbone shows that 40 ms and 80 ms are convenient values especially if redundant techniques [28] are used to minimize the effect of large "holes" in the speech. 80 ms appears to be a good compromise between interactivity needs and bandwidth overhead generated by the hierarchical coding.

In some cases (e.g. very low bandwidth links), the IP/UDP/RTP headers may be compressed to reduce the bandwidth overhead [8]. However, such techniques increase the CPU consumption at the routers and they are expected to be used on point to point links, but not on an end-to-end basis on the Mbone.

Speech Coding/ Transmission	Payload (kbps)	20ms buffer		40ms buffer		80ms buffer	
		Total (kbps) IPv4/IPv6	Efficiency ($\frac{\text{payload}}{\text{header}}$) IPv4/IPv6	Total (kbps) IPv4/IPv6	Efficiency ($\frac{\text{payload}}{\text{header}}$) IPv4/IPv6	Total (kbps) IPv4/IPv6	Efficiency ($\frac{\text{payload}}{\text{header}}$) IPv4/IPv6
PCM	64	81.6/88.0	3.64/2.67	72.8/76.0	7.27/5.33	68.4/70.0	14.55/10.67
ADM4	32	50.1/56.5	1.82/1.33	40.8/44.0	3.64/2.67	36.4/38.0	7.27/5.33
PCM 1 fl. 2.7kHz	21.3	38.9/45.3	1.21/0.89	30.1/33.3	2.42/1.77	25.7/27.3	4.84/3.55
PCM 3 fl. 2.7kHz	64	117/136	1.21/0.89	90.3/100	2.42/1.77	77.7/81.9	4.84/3.55
ADM4 1 fl. 2.7kHz	10.7	28.3/34.7	0.61/0.45	19.5/22.7	1.22/0.89	15.1/16.7	2.43/1.78
ADM4 3 fl. 2.7kHz	32	84.9/104	0.61/0.44	58.5/68.1	1.22/0.89	45.3/50.1	2.43/1.78
Overhead (1 fl.)	N/A	17.6/24.0	N/A	8.8/12.0	N/A	4.4/6.0	N/A
Overhead (3 fl.)	N/A	52.8/72.0	N/A	26.4/36.0	N/A	13.2/18.0	N/A
Overhead (6 fl.)	N/A	106/144	N/A	52.8/72.0	N/A	26.4/36.0	N/A

Table 1: Bandwidth overhead for several audio coding/transmission schemes

Overhead per packet (in bytes)	IP		UDP	RTP
	v4	v6		
	24	40	8	12

Table 2: IP/UDP/RTP header size

2.4 Implementation details

The hierarchical encoded flows are carried as payload data within the RTP protocol [32]. The application follows the recommendations defined in the "RTP usage with layered multimedia streams" draft [35]. There is no need to manage a different sequence numbering per flow. So, in our hierarchical audio application, we further impose that all flows use the same sequence number to encode an audio sample. The receiver handles one circular buffer for all the layers, so only one playout is computed using the algorithms described in [29].

3 The receiver-based control scheme

3.1 The RLM scheme

Receiver driven Layered Multicast (RLM) is the first published scheme which described a specific multicast layered transmission scheme and the associated receiver control scheme. RLM uses "probing" experiments similar to those used by TCP to decide when to join and quit layers. Specifically, when a receiver detects congestion, it quits the multicast group corresponding to the highest layer it is receiving at the time (we say that the receiver drops the highest layer); when a receiver detects spare capacity in the network, it joins the multicast group corresponding to the layer next to the highest layer received at the time (we say that the receiver adds the next layer).

The receiver detects network congestion when it observes increasing packet losses. In the absence of loss, the receiver estimates spare capacity, or rather the existence of spare capacity, with so-called join experiments. A join experiment means that a receiver joins the next group and measures the loss rate over an interval referred to as the decision time (to avoid the synchronization of join experiments, experiments are carried out at randomized times). However, the load created by join experiments increases as the size of the multicast group increases. To prevent a load explosion, the authors in [22] use "shared learning", in which a receiver about to start a join experiments multicast its intent to the group. Thus all receivers get to know the result of join experiments carried out by other receivers. To prevent join experiment for different layers to interfere with each others, only receivers that are doing join experiments for layers equal or below a newly advertised experiment can actually conduct their own experiments. RLM with shared learning scales with the size of the group, but it raises a few questions. In particular, it is not clear how effective shared learning is in the absence of knowledge about the structure of the multicast delivery tree. Furthermore, shared learning increases the convergence time to steady state layer subscription especially for receivers with spare capacity (since they will have to wait for slow receivers to reach their steady state before they can join additional upper layers). Also, the times at which layers are added and dropped determine the rate increase and decrease along the branches of the tree. Thus, RLM should choose these times appropriately if it wants to achieve fairness with TCP traffic. However, this appears hard to do in practice.

3.2 Basic idea

Our control scheme aims at providing the benefits of the RLM scheme without (at least some of) the costs associated with it. The scheme uses the same idea of a receiver based rate control scheme, however join experiments are replaced by an explicit estimation at each receiver of the bandwidth that would be used by an equivalent TCP connection between the source and the receiver. This estimation is done using the same technique as that recently described to design TCP-friendly unicast rate control scheme [20]. Specifically, the

bandwidth λ_{equ} of a TCP connection can be well represented by Equation 1 below [20, 21, 25]

$$\lambda_{equ} = 1.22 * \frac{MTU}{RTT * \sqrt{Loss}} \quad (1)$$

where MTU is the maximum packet size used on the connection, RTT is the mean round trip time, and $Loss$ is the mean packet loss rate. Now assume that each receiver knows the rate λ_i of the layer i flow generated by the source over the multicast tree. Then, each receiver executes the following algorithm:

Step 0: Upon joining the group, subscribe to the base layer

Step 1: Measure or estimate MTU , RTT , and $Loss$

Step 2: Compute λ_{equ}

Step 3: Find L , the largest integer such that $\sum_{i=1}^L \lambda_i \leq \lambda_{equ}$. Join the first L groups.

Note that the rate at which data flows between the source and any receiver is equivalent to that of a TCP connection running over the same path. Thus, we have obtained a TCP-friendly scheme suitable for multicast delivery. Furthermore, note that the scheme does not rely on active probing schemes such as join experiments, nor does it require exchange of information between participants as is done with shared learning. We examine below details associated with steps 1 and 3 above (e.g. the estimation of parameters in step 1). However, before doing this, it is worth going back to Equation 1 and note that i) the equation in fact only provides an *upper bound* to the rate of an equivalent TCP connection, and ii) however it has been shown to fit well with measured and simulated performance of a wide variety of TCP variants [13].

3.3 Parameter estimation

Step 1 in the list above deals with estimating network parameters. So let us consider all of them in turn.

MTU: The MTU value may be set to the the minimum acceptable value for TCP of 576 bytes or could be determined using the MTU discovery algorithm [24].

Loss: The mean loss rate $Loss$ is easily computed from observed losses (detected using the RTP sequence number field) using an exponential filter with a half life equal to n packets (or equivalently to $d * n$ sec, where d denotes the inter-packet arrival). The trick of course is to choose an appropriate value for n . We use a value that varies over time as shown in [22].

RTT: The mean round trip delay RTT cannot be computed as easily as $Loss$ since the information received from the source in RTP SRs (*Sender Reports*) only refers to the one-way delay between source and receiver. Unfortunately, the one way delay can be a poor estimate

of $RTT/2$ [6, 26]. There are 3 ways to tackle this problem. The first way is to estimate RTT using twice the delay from source to receiver (which is known using RTP timestamps), and assume a symmetric path. However, this clearly breaks down, in particular with asymmetric links (such as satellite links). The second way is to augment the capabilities of RTP so as to be able to estimate the actual round trip delay. this can be done as follows: the receiver sends an rtt-request packet; the sender replies immediately with an rtt-reply packet to the receiver; the receiver notes the delay corresponding to the current RTT value and updates the RTT estimator with it. Unfortunately, this scheme does not scale to large multicast sessions. First, the frequency at which the receiver generates the rtt-request packets should be function of the session size (as with RTCP messages [32]) in order to avoid the well-known feedback implosion problem. Second, the period between two rtt-request packets should not be constant in order to avoid RTT synchronized requests and periodic congestion [12]. Third, unicast rtt-requests from the sender may be less efficient than multicast joint rtt-requests. We could have used the SR and RR (*Receiver Report*) packets to append the information required to estimate RTT . However, these packets include several parameters that do not need to be sent as often than the RTT information. So, in order to save bandwidth, we have created two new RTCP packets which include the minimal information needed to estimate RTT : *rttreq* for the rtt-request packets and *rttrep* for the rtt-reply packets sent by the sender. *rttreq* packets include a 4-bytes RTCP header which identifies the packet and encodes the packet length, the SSRC of the receiver, and the time at which the rtt-request has been sent at the receiver t_{s_rttreq} . *rttrep* packets include the 4-bytes RTCP header, the SSRC of the sender and a set of RTT information report blocks corresponding to each *rttreq* packets received since the last *rttrep* report. Each RTT information report block includes SSRC of the receiver, the corresponding t_{s_rttreq} parameter and the delay δ in millisecond between the time at which the *rttreq* has been received at the sender and the time at which the *rttrep* is sent to the group, see Figure 3.

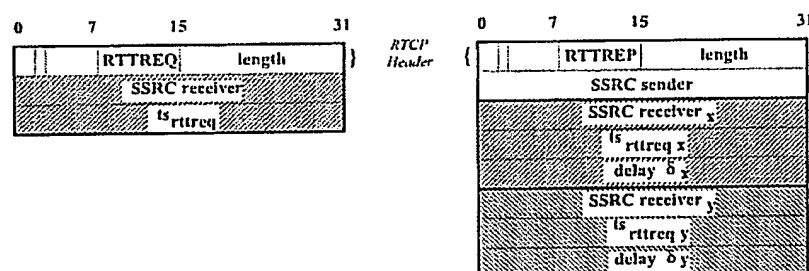


Figure 3: rttreq and rttrep packets

As soon as a receiver receives an *rttreq* report, it notes the corresponding receiving time t_{r_rttreq} . Then, if its SSRC is included in the packet, it is able to update the estimation of

the RTT with the new value:

$$RTT = tr_{rttrep} - ts_{rttreq} - \delta \quad (2)$$

Note that this mechanism does not assume that sender and receivers clocks are synchronized.

Let us now examine how this parameter estimation mechanism scales with the number of receivers N in the session. To do this, we compute next the interval of time T_{RTT} between two RTT estimations for a receiver in the session. Assume that r % of the payload bandwidth BP is reserved for this usage.

$$BP_{RTT} = \frac{W_{rttrep} + N.W_{rttreq}}{T_{RTT}} \quad (3)$$

with W_{rttrep} , the size of the $rttrep$ packet and W_{rttreq} , the size of the $rttreq$ packet. From Figure 3, we have $W_{rttrep} = 64 + 96.N$ bits and $W_{rttreq} = 96$ bits. So, using equation 3:

$$T_{RTT} = \frac{64 + 192.N}{BP_{RTT}} \quad (4)$$

Figure 4 shows the delay between two RTT updates with N in $[1, 200]$ with the following parameters: $BP_{RTT} = 64kb/s$, $r = 5\%$. With $N = 16$ receivers, the RTT estimate is

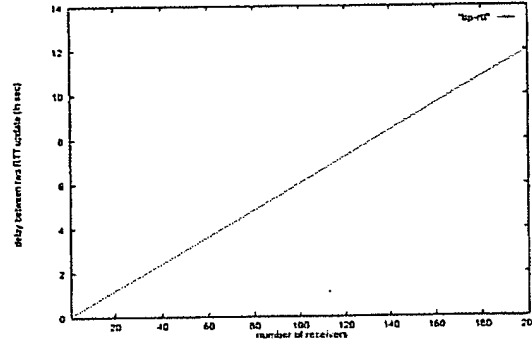


Figure 4: Delay between two RTT estimations (in sec) vs N

updated about each second. When the number of receivers reaches 100, the interval of time increases to 6 seconds. Clearly our scheme is not convenient for large groups since it suffers from the same scaling problems than RTCP [31]. All the more so since some of the $rttreq$ and $rttrep$ packets can be lost during transmission.

This has lead us to examine a third way to estimate RTT , which in fact attempts to bypass RTT entirely in Equation 1. The goal there is to obtain a relation for λ_{equ} in ways similar to those described in [25, 19] that does not involve RTT , but rather the source to destination delay and the inter-packet jitter observed at the destination. The idea is to

measure *RTT* infrequently (e.g using the scheme above), then to track variations in the one-way source-destination delay using RTP timestamps, and to track variations in the one-way destination-source delay using the jitter at the destination. Indeed, variations in the destination-source delay are translated in the source variations in the times at which packets are sent and thus into jitter at the destination. Unfortunately, we have not been able to derive an approach formula for λ_{equ} yet, and this remains an area for future research.

A fourth approach, which we only mention, is to forget about TCP-friendliness altogether and instead wait for schemes such as RED to be deployed in the Internet....

3.4 Control algorithm

At the beginning, when a receiver joins a session, it subscribes to the base layer in order to estimate the parameters as described above during a time interval T_i . Once parameters have been estimated, Equation 1 is used to estimate the rate (λ_{equ}) of an equivalent TCP connection. The value of λ_{equ} in turn indicates how many layers the algorithm may subscribe to so as to behave (at least throughput wise) like a TCP connection. At the initialization, the T_i time interval is set to the length corresponding to the exponential filters (for *RTT* and *Loss* parameters). Then, we ensure that the T_i interval be at least greater than the mean *RTT* value estimated in order that the receiver can detect the resulting impact of the latest action performed.

A receiver actually subscribes to extra multicast group(s) when it is allowed to do so. Once this is done, it resumes the observation and estimation of the parameters so as to determine the impact of the join on network congestion. The receiver won't take any actions until it can estimate the effect of its latest action. This period of time equals to the maximum between the estimated *RTT* and the delay corresponding to the filter length.

The situation is similar, in the opposite direction, when the equivalent rate tells the receiver to drop a layer. Finally, if the equivalent rate is less than the minimal rate the application can send (i.e. the base band rate), the algorithm should pop up a message to the receiver and invite (or force) him to leave the session in order not to congestion the network and swamp all the network resources. – Internet collapse could happen for example if a large number of receivers continue to subscribe to the minimal quality flow (i.e. no more rate decrease in case of high packet loss observed). To follow the behavior of TCP [18], the application can wait for a delay T_r before probing the network state by joining the first layer. $T_r = 2^k + \sigma$, where k is the number of consecutive failed attempts, and σ a random value added to avoid synchronization of receivers.

4 Experimental evaluation over the MBone

The layer codec and the control which described above have been implemented in an experimental version of the FreePhone audio tool [15]. Packets are sent using the usual IP/UDP/RTP stack. All the measurements results presented in the paper have been done with a hierarchical and non-hierarchical ADM4 coder. Audio packets sent by the coder

Table 3 shows the values of *RTT* (in ms) and *Loss* (in %) corresponding to the 3 receivers. We note that in the second experiment, the values of *RTT* and *Loss* measured at a receiver are essentially constant. However, in the first experiment, and especially when the network is loaded, *RTT* and *Loss* increase with the flow number⁵. To explain this phenomenon, we can use the cumulated loss (CL) rate parameter which represents the probability to loose all the packets (from different flows) corresponding to a sample. The cumulated loss rate is computed at the receiver side, after reconstructing all samples with packets received from all the flows. The results are shown in Table 3 for the two experiments. The cumulating loss rate for the 3 flows received is about eight times lower in the first experiment than in the second one. This means that the higher packet loss rate observed in the second experiment is mainly due to the loss of consecutive packets sent. The low value obtained in the first experiment (less than 0.5 %) clearly shows that this hierarchical coding scheme uses efficiently each flow as a redundancy for the others flows received. We have plotted in Figures 6, 7 and 8, the evolution of the cumulating loss rate vs the sequence number (or sample number) corresponding respectively to INRIA, UCL and UTS receivers. The left and the right figures show the cumulating packet loss rate when the packets are sent respectively per burst and at regular time interval, i.e. with *shift* option enabled. We can observe that at each receiver, the probability to loose 3 consecutive packets (i.e. an sample sent in 3 flows) is very low.

So the sender's application needs to include a mechanism to avoid sending bursts of packets in different flows (in the same way an efficient non-hierarchical sender's application acts). Note that the periodical packet loss observed at UTS (Figure 8 shows three consecutive packet loss roughly every 370 packets, which corresponds to 30 seconds) is likely due to the router's synchronism bug in which routers periodically exchange their routing table over the network [12].

Table 3 also shows that by using the *shift*, the mean *RTT* value and the mean packet loss rate are nearly constant for the three flows. This makes a nice point about the scalability of the scheme since we can choose a single flow to estimate the parameters for the receiver's congestion control algorithm. In the rest of the paper, the estimation of the *RTT* and packet loss rate parameters is done on the base flow and experimentations are done using the *shift* option enabled.

4.2 Evaluation of the congestion control algorithm

Once we have studied the correlations of flows on the MBone, we can evaluate the congestion control algorithm described in section 3.

In a third experiment, the source at MIT sends 5 flows corresponding to the hierarchical audio coding rate described in section 2. The compression algorithm selected is ADM4 and the audio flow is originally at 16 kHz (the output flows corresponds to two ADM4 / 8 kHz audio see Table 1.). The receivers implement the congestion control algorithm described in section 3 to decide how many flows they are allowed to subscribe to.

⁵ We have noticed that the difference increases with the network load.

Receiver	flow #	with shift			without shift		
		RTT (ms)	loss (%)	CL (%)	RTT (ms)	loss (%)	CL (%)
INRIA	1	301	6.0	6.0	306	5.9	5.9
	2	301	5.9	1.2	309	7.5	3.6
	3	301	6.3	0.4	313	9.0	3.1
UCL	1	357	4.4	4.4	360	3.9	3.9
	2	358	4.1	0.3	363	4.3	1.2
	3	358	5.0	0.1	367	4.6	0.8
UTS	1	308	0.9	0.9	308	1.1	1.1
	2	307	0.9	0.2	312	1.1	0.4
	3	308	0.7	0.1	312	0.9	0.3

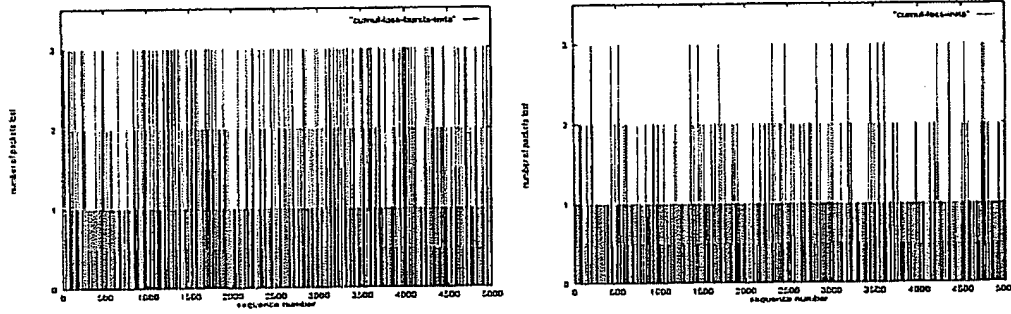
Table 3: *RTT*, mean and cumulated packet loss intercorrelation between flows

Figure 6: Cumul loss rate versus sequence number at INRIA

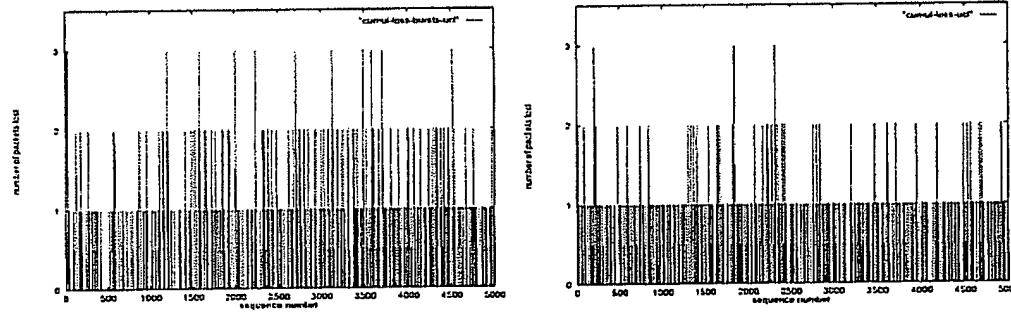


Figure 7: Cumul loss rate versus sequence number at UCL

INRIA

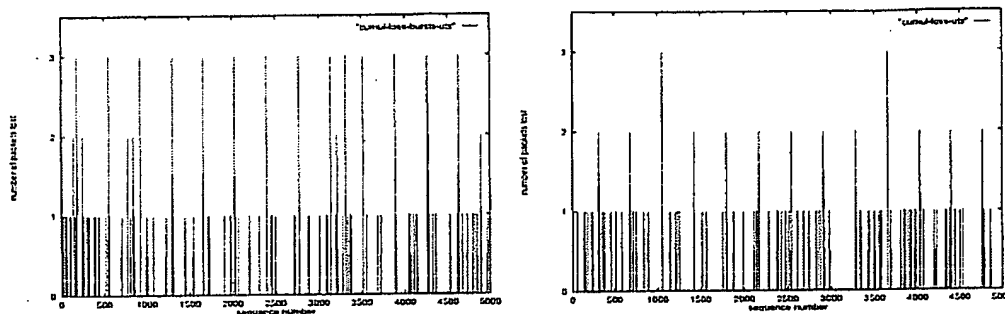


Figure 8: Cumul loss rate versus sequence number at UTS

Figures 9 and 10 show the evolutions of the RTT and the packet loss rate estimators respectively according to the sequence number (corresponding to the audio sample number) for both UTS and UCL receivers. The corresponding maximal rate estimation allowed and the subscription decision is plotted in Figure 11. The results from the local receiver at MIT have not been plotted but show as expected a very low RTT value and a null packet loss rate which set the level of flow subscription to its maximal value (i.e. 5).

As expected, we observe that during periods of congestion (with higher RTT and packet loss rate estimates), the algorithm selects a lower number of flows. When the network is unloaded, and as soon as the first estimations of the RTT and the packet loss rate are available, the algorithm can select the higher rate it is allowed to receive (e.g. for the UCL's receiver). The algorithm can also decide to drop several flows at the same time in case of severe congestion.

In the fourth experiment, we examine how multiple sessions interact while running in a same network. We have run two different multicast sessions over a low-bandwidth PPP link⁴ set between two PCs and used only by these sessions. The two sessions use an ADM4 (32 kb/s payload) coding at 8 kHz sent over 3 layers with 80 ms of speech encoded per packet. Figure 12 shows the subscription decision and the rate received for the two different sessions. We note that both sessions have roughly the same behaviour, i.e. the subscription decision oscillates between 1 and 2 flows and the mean bandwidth received is about 12 kb/s for session 1 and 13 kb/s for session 2. In case of several sessions are run under the same network conditions (i.e. same packet loss and RTT), we observe that each session gets roughly the same amount of the total network capacities. During tests, we have also noticed that the PPP link is more likely to loose bursts of packets rather than isolated packets such as common in the MBone. Since the measure of quality of the coding depends on the packet loss properties in the network, the following tests have been done on the MBone, even if we cannot ensure the same network conditions for two consecutive experiments.

⁴ The maximal UDP bandwidth measured over this low-rate link was 83 kb/s using a packet size equal to the MTU=1024 bytes.

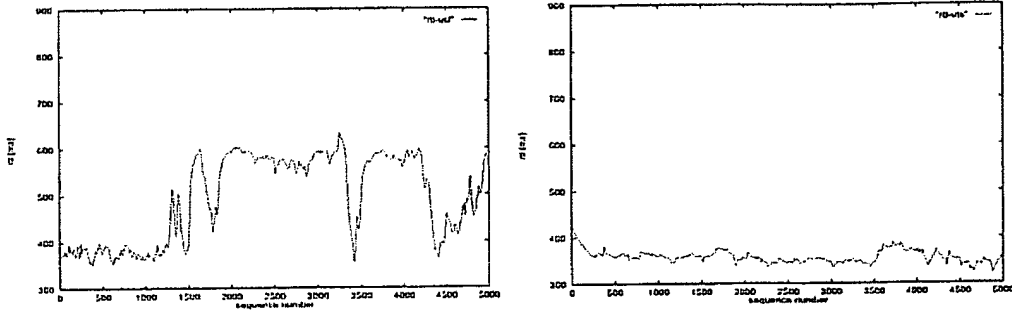
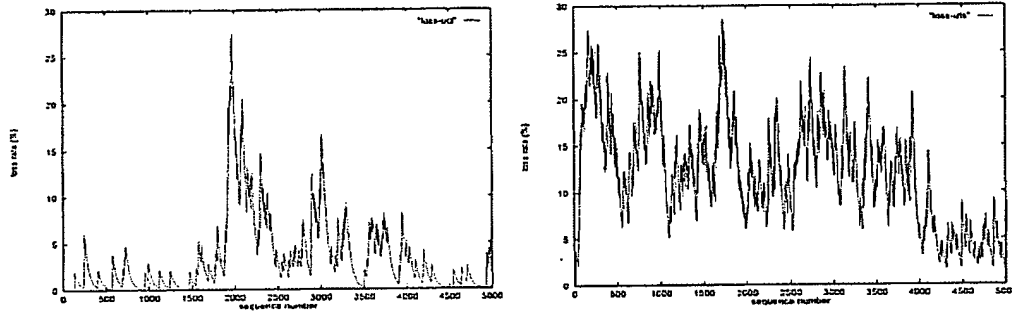
Figure 9: *RTT* versus sequence number at UCL (left) and UTS (right)

Figure 10: Loss rate versus sequence number at UCL (left) and UTS (right)

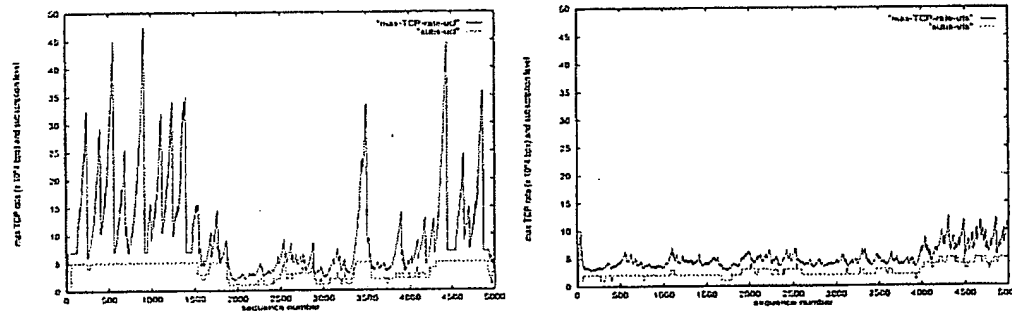


Figure 11: Subscription level versus sequence number at UCL (left) and UTS (right)

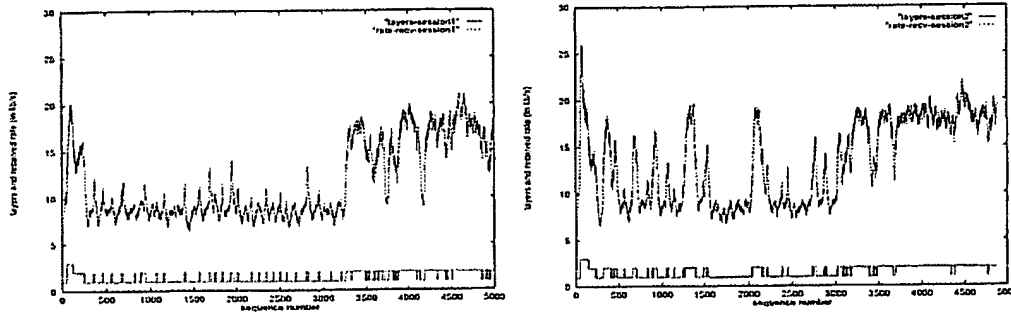


Figure 12: Flows subscribed and Received rate for two different sessions over PPP

4.3 Comparison with “traditional” schemes

It is interesting to compare our scheme with the transmission scheme used in “traditional” applications such as vat. In the fifth experiment, the source at MIT sends the equivalent ADM4 compressed 16 kHz data in one flow. This experiment has been done a couple of minutes after the third one, and not in the same time to limit our sending rate on the MBone. Of course, network conditions can be slightly different between two experiments, so we cannot simply use the rates observed in both experiments to compare quality obtained.

Using this layered coding scheme in conjunction with a rate control transmission algorithm will achieve a variable audio quality according to the network conditions. Further tests are needed with real users in order to observe their behavior during an audio conference. During a session the audio quality may sometimes appear “metallic” when only one flow is received and may sometimes be HiFi (e.g. all flows received corresponding to a 32 kHz / PCM audio flow). The best way to measure the quality of the audio flows received would have been to use a MOS measurement [34]. However, there is no such measure available yet for the experimental PCM and ADM hierarchical encoded schemes described in section 2. To coarsely quantify the quality obtained at receivers, we use the instantaneous rate received for each sequence number. The quality of the audio signal increases with the number of packets received by sample. The instantaneous quality is null when the sample cannot be reconstructed and is maximal when all the packets constituting the sample are received.

Figures 13 and 14 show respectively the evolutions of the packet loss rate and the corresponding quality observed at UCL and UTS for the non-hierarchical sending experiment (fifth experiment).

We note that in the fifth experiment, the quality received instantaneously is either minimal (i.e. sample lost) or maximal (sample totally reconstructed). Some of the receivers (e.g. UTS) receive lots of “blanks” (e.g. 1 loss encodes 80ms, 5 consecutive losses 0.4 second) which make the audio flow sometimes unintelligible. Audio samples corresponding to several experiments with various packet loss rates can be retrieved at <http://www.inria.fr/rodeo/turletti/audio/> so as to be able to listen to the quality of our scheme versus “traditional schemes”. Moreover,

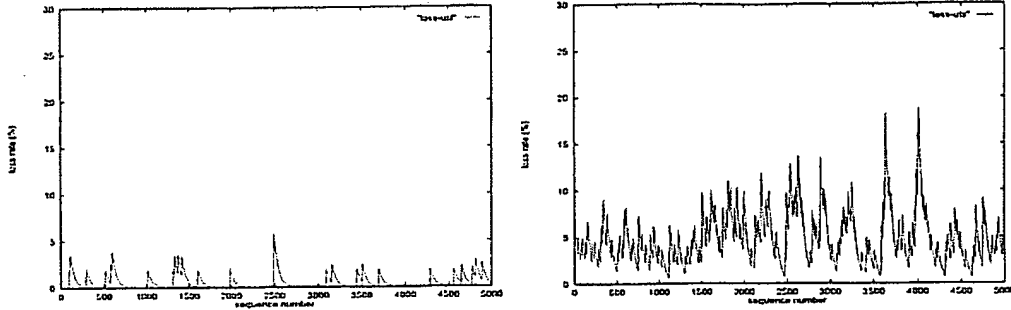


Figure 13: Packet loss rate for non hierarchical flows at UCL (left) and UTS (right)

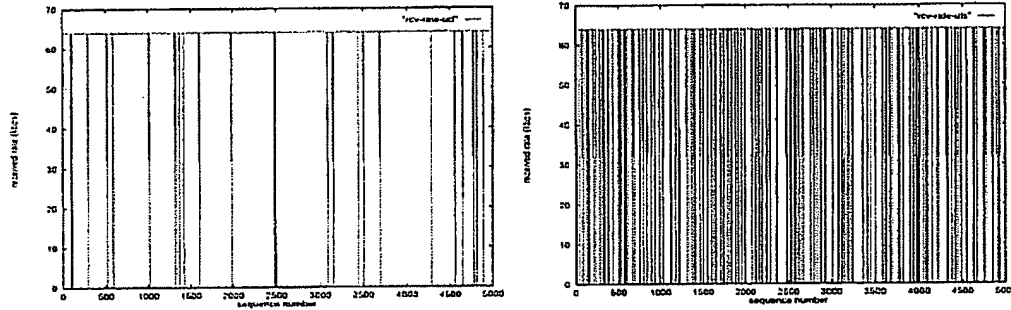


Figure 14: Received rate for non hierarchical at UCL (left) and UTS (right)

the output rate sent in the network does not back off and is always set to its maximal value (i.e. 64 kbps), yielding a "bad citizen" behavior which may contribute to congestion in the Internet.

Now let us examine how our layered transmission scheme behaves in presence of packet loss. Figures 15 and 17 show respectively the evolutions of the packet loss rate and the corresponding quality observed at UCL and UTS for the third experiment.

In period of high congestion (when the receiver subscribes to only one flow), the robustness of this hierarchical coding scheme allows to reconstruct all the packet lost at UCL and UTS. Clearly, this will not be the case if congestion is yet more severe, but in this case, the maximal subscription rate allowed could overrun the minimal sending rate of the application (e.g. here 21 kb/s of payload) and the receiver should be invited to quit the session and restart later.

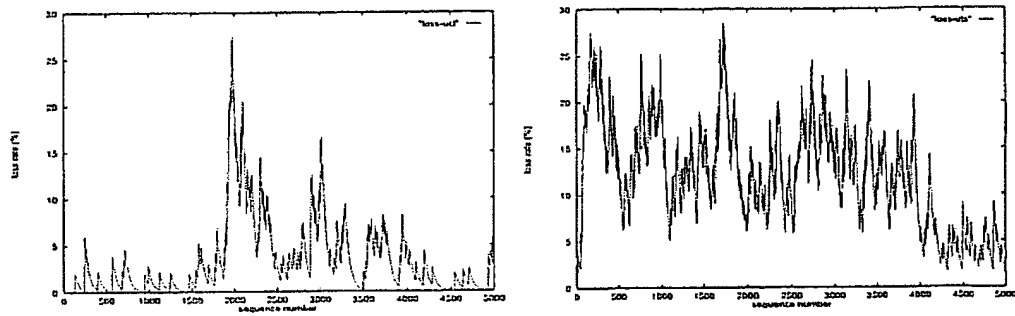


Figure 15: Packet loss rate for hierarchical flows at UCL (left) and UTS (right)

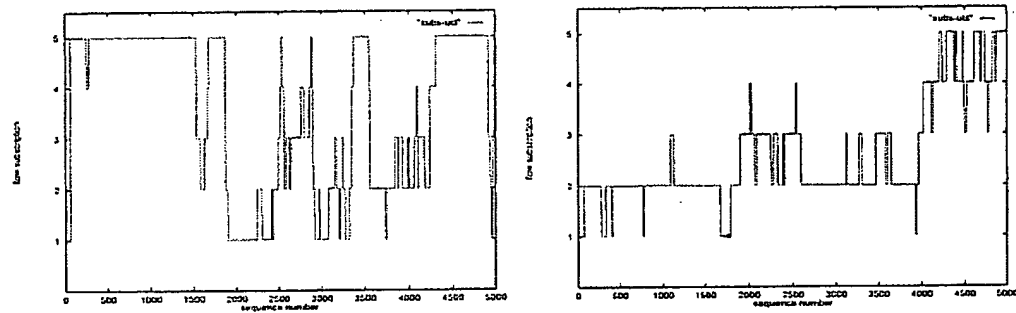


Figure 16: Subscription level for hierarchical flows at UCL (left) and UTS (right)

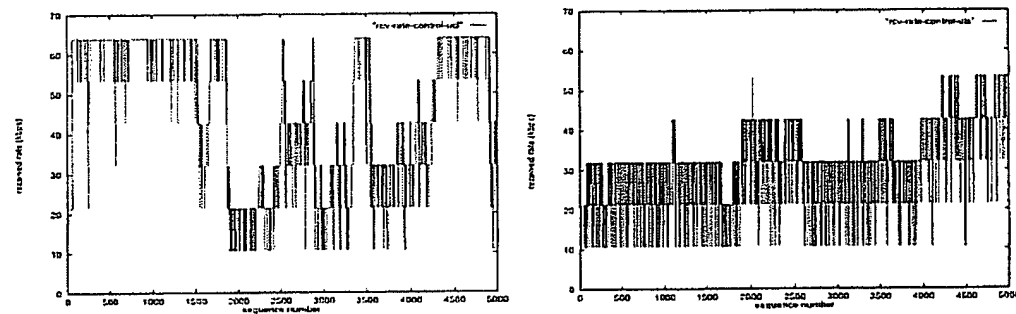


Figure 17: Received rate for hierarchical flows at UCL (left) and UTS (right)

5 Conclusion

The control scheme presented in the paper has a number of interesting features compared to RLM. In particular, it has "built-in" TCP friendliness, it does not require coordination between receivers, and we have seen that results from an actual implementation show good convergence properties (in addition to the expected result that the combination of a layered coding and a layered transmission scheme does help to handle the problem of multicast delivery of heterogeneous networks). However, there remain a number of problems, most of them actually related to Equation 1. First, the equation only provides an upper bound on the rate of the equivalent TCP connection; there is a need to investigate how far the bound is to the actual value. Second, it is not yet clear exactly how a rate based control scheme based on equation 1 and an actual window based TCP scheme interact in practice. Third, the equation relies on parameters such as the mean roundtrip delay RTT that are problematic to estimate accurately in large multicast environments. This, combined with questions about the relevance of TCP-friendliness in networks with RED gateways, means that the jury is still out on receiver based control schemes for multicast environments, and that this remains an important (and somewhat urgent) area for research.

References

- [1] E. Amir, S. McCanne, H. Zhang, "An Application-level Video Gateway", *Proc. ACM Multimedia '95*, San Francisco, Nov. 1995.
- [2] Archive of the rem-conf mailing list at <ftp://nic.es.net/pub/mailling-lists/mail-archive/rem-conf>
- [3] J-C. Bolot, T. Turetti, I. Wakeman, "Scalable feedback control for multicast video distribution in the Internet", *Proc. ACM/SIGCOMM'94*, Vol. 24, No 4, pp. 58-67, October 1994.
- [4] S. Casner, "First IETF Internet audiocast", *Computer Communication Review*, vol. 22, no. 3, pp. 92-97, July 1992.
- [5] N. Chaddah, "Software only scalable video delivery system for multimedia applications over heterogeneous networks", *Proc. ICIP 95*, Wash. DC, Oct. 1995.
- [6] K. Claffy, H.-W. Braun, G. Polyzos, "Measurement considerations for assessing unidirectional latencies", *Journal of Internetworking*, vol. 4, no. 3, September 1993.
- [7] S. Y. Cheung, M. H. Ammar, X. Li, "On the Use of Destination Set Grouping to Improve Fairness in Multicast Video Distribution", *Proc. IEEE Infocom '96*, San Fransisco, CA, pp. 553-560, Apr. 1996.
- [8] S. Casner, V. Jacobson, "Compressing IP/UDP/RTP Headers for Low-Speed Serial Links", *Internet-Draft*, Jul. 14, 1997.
- [9] CNET reviews:
 Internet phones: <http://www.cnet.com/Content/Reviews/Compare/Webphone/>
 Videoconferencing tools: <http://www.cnet.com/Content/Reviews/Compare/Video/>
 Collaborative tools: <http://www.cnet.com/Content/Reviews/Compare/Netconference/>

- [10] J. Danskin, G. Davis, X. Song, "Fast lossy Internet image transmission", *Proc. ACM Multimedia '95*, Boston, MA, pp. 321-332, Nov. 1995.
- [11] B.J. Dempsey, J. Liebeherr, and A.C. Weaver, "On Retransmission-Based Error Control for Continuous Media Traffic in Packet-Switching Networks", *Computer Networks and ISDN Systems*, Vol. 28, No. 5, pp. 719 - 736, March 1996.
- [12] S. Floyd, V. Jacobson, "The synchronization of periodic routing messages", *Proc. ACM Sigcomm '93*, San-Francisco, CA, pp. 33-44, Sep. 1993.
- [13] S. Floyd, K. Fall, "Router Mechanisms to Support End-to-End Congestion Control" *Technical Report*, Feb. 15, 1997.
- [14] A.V. Garcia, "Control Mechanisms for Audio Transmission over the Internet" *PhD Thesis*, October 96.
- [15] Free Phone, <http://zenon.inria.fr/rodeo/fphone/>
- [16] M.W. Garrett, M. Vetterli, "Joint source/channel coding of statistically multiplexed real time services on packet networks", *ACM/IEEE Trans. Networking*, vol. 1, no. 1, pp. 71-80, Feb. 1993.
- [17] Internet Telephony Consortium (ITC), <http://itel.mit.edu/>
- [18] V. Jacobson, "Congestion avoidance and control", *Proc. ACM Sigcomm '88*, Stanford, CA, pp. 314-329, Aug. 1988.
- [19] T.V. Lakshman, U. Madhow, B. Suter, "Window-based error recovery and flow control with a slow acknowledgment channel: a study of TCP/IP performance", *IEEE Infocom'97*, Kobe, Japan, 1997.
- [20] J. Mahdavi, S. Floyd, "TCP-Friendly Unicast Rate-Based Flow Control" *Draft posted on end2end mailing list*, Jan. 8, 1997.
http://www.psc.edu/networking/papers/tcp_friendly.html
- [21] M. Mathis, J. Semke, J. Mahdavi, T. Ott, "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm" To appear in *Computer Communication Review*, Vol. 27, No 3, July 1997.
- [22] S. McCanne, V. Jacobson, M. Vetterli, "Receiver-driven Layered Multicast", *Proc. ACM Sigcomm '96*, Stanford, CA, pp. 117-130, Sept. 1996.
- [23] S. McCanne, M. Vetterli, V. Jacobson, "Low-complexity Video Coding for Receiver-driven Layered Multicast", to appear in *IEEE JSAC '97*.
- [24] J. Mogul, S. Deering, "Path MTU Discovery", RFC 1063, Nov. 1990.
- [25] S. Ott, J. Kemperman, M. Mathis, "Window Size behavior in TCP/IP with Constant Loss Probability", *DIMACS Workshop on Performance of Realtime Applications on the Internet*, Plainfield NJ, Nov. 6-8, 1996.
- [26] V. Paxson, "Measurements and analysis of end-to-end Internet dynamics", *PhD Thesis*, University of California, Berkeley, Apr. 97.
- [27] J. Pasquale et al., "Filter propagation in dissemination trees: Trading off bandwidth and processing in continuous media networks", *Proc. NOSSDAV '93*, Lancaster, UK, pp. 269-278, Oct. 1993.

- [28] C. Perkins et al, "RTP Payload for Redondant Audio Data", *Internet-Draft*, Jan. 3, 1997, expires June 1997.
- [29] R. Ramjee, J. Kurose, D. Towsley, H. Schulzrinne, "Adaptive playout mechanisms for packetized audio applications in wide-area networks", *Proc. IEEE Infocom '94*, Toronto, Canada, pp. 680-688, April 1994.
- [30] Robust-Audio Tool (RAT), <http://www-mice.cs.ucl.ac.uk/mice/rat/>
- [31] J. Rosenberg, H. Schulzrinne, "Scaling Feedback to Very Large Groups" *Research report available via <http://www.cs.columbia.edu/hgs/papers/Rose9701:Scaling.ps.gz>*, Feb. 13 1997.
- [32] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, "RTP: A transport protocol for real-time applications", *RFC 1889*.
- [33] N. Shacham, P. McKenney, "Packet recovery in high-speed networks using coding and buffer management" *Proc. IEEE Infocom '90*, San Francisco, CA, pp. 124-131, June 1990.
- [34] A. Spanias, "Speech coding: A tutorial review", *proc. IEEE*, vol. 82, no. 10, pp. 1541-1582, Oct. 1994.
- [35] M.F. Speer, S. McCanne, "RTP usage with layered multimedia streams", *Internet-Draft*, June 1, 1996, expires Dec. 1996.
- [36] LBNL Audio Conferencing Tool (VAT), <http://www-nrg.ee.lbl.gov/vat/>
- [37] VocalTec <http://www.vocaltec.com/>
- [38] Voice On the Net (VON), <http://www.von.com/>
- [39] C. Weinstein, J. Forgie, "Experience with speech communication in packet networks", *IEEE JSAC*, vol. 1, no. 6, pp. 963-980, Dec. 1983.

A The receiver-based control pseudo code

```

...
int N;
int NbFlow = 0;
int k = 0;
ACTION = JOIN;
...

// Receive Data and/or Control Packet

void Packet_Received()
{
    if (Rtp_Packet()) {
        SeqNb = Get_Sequence_Number();
        Update_Loss();
        Compute_TCP_Rate();
        Cpt_RTT -= 1;
        if (! Cpt_RTT) {
            Cpt_RTT = N;
        }
    }
}

```

```

        Send_RR_Packet(SeqNb, CurrentTime());
    }
} else
    if ((Rtcp_Packet()) && (Is_SR_Packet()) &&
        (Packet_Ssrc() == MySSRC)) {
        UpdateRTT();
        Compute_TCP_Rate();
    }
}

// Rate Control Algorithm

void Algo(...)
{
    switch(ACTION) {
        case JOIN:
            if (NbFlow)
                k = 0;
            L = Nb_Of_Layer(TCP_Rate(), Our_Rate());
            Join_Layers(L, &NbFlow);
            Wait(n*d);
            ACTION = NONE;
            break;
        case QUIT:
            L = Nb_Of_Layer(TCP_Rate(), Our_Rate());
            Quit_Layers(L, &NbFlow);
            if (NbFlow >= 1) {
                Wait(n*d);
                ACTION = NONE;
            } else
                ACTION = RETRY;
            break;
        case NONE:
            if (TCP_Rate() > Our_Rate()+NEXT_Layer_Rate())
                ACTION = JOIN;
            else
                if (TCP_RATE() < Our_Rate())
                    ACTION = QUIT;
                else
                {
                    ACTION = NONE;
                    k=0;
                }
            break;
        case RETRY:
            tr = power(2, ++k);
    }
}

```

```
        wait(tr);  
        Join_Layers(1, &NbFlow);  
        Wait(n*d);  
        ACTION = NONE;  
        break;  
    }  
    ...  
}
```




Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - B.P. 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Lorraine : Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - B.P. 101 - 54602 Villers lès Nancy Cedex (France)

Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot St Martin (France)

Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, B.P. 105 - 78153 Le Chesnay Cedex (France)
<http://www.inria.fr>
ISSN 0249-6399

The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm

Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi *
<mathis@psc.edu> <semke@psc.edu> <mahdavi@psc.edu>
Pittsburgh Supercomputing Center

Teunis Ott
<tjo@bellcore.com>
Bellcore

Abstract

In this paper, we analyze a performance model for the TCP Congestion Avoidance algorithm. The model predicts the bandwidth of a sustained TCP connection subjected to light to moderate packet losses, such as loss caused by network congestion. It assumes that TCP avoids retransmission timeouts and always has sufficient receiver window and sender data. The model predicts the Congestion Avoidance performance of nearly all TCP implementations under restricted conditions and of TCP with Selective Acknowledgements over a much wider range of Internet conditions.

We verify the model through both simulation and live Internet measurements. The simulations test several TCP implementations under a range of loss conditions and in environments with both drop-tail and RED queuing. The model is also compared to live Internet measurements using the TReno diagnostic and real TCP implementations.

We also present several applications of the model to problems of bandwidth allocation in the Internet. We use the model to analyze networks with multiple congested gateways; this analysis shows strong agreement with prior work in this area. Finally, we present several important implications about the behavior of the Internet in the presence of high load from diverse user communities.

1 Introduction

Traffic dynamics in the Internet are heavily influenced by the behavior of the TCP Congestion Avoidance algorithm [Jac88a, Ste97]. This paper investigates an analytical performance model for this algorithm. The model predicts end-to-end TCP performance from properties of the underlying IP path. This paper is a first step at discovering the relationship between end-to-end application performance, as observed by an Internet user, and hop-by-hop IP performance, as might be monitored and marketed by an Internet Service Provider.

*This work is supported in part by National Science Foundation Grant No. NCR-9415552.

Our initial inspiration for this work was the "heuristic analysis" by Sally Floyd [Flo91].

This paper follows a first principles derivation of the stationary distribution of the congestion window of ideal TCP Congestion Avoidance subject to independent congestion signals with constant probability. The derivation, by Teunis Ott, was presented at DIMACS [OKM96b] and is available on line [OKM96a]. The full derivation and formal analysis is quite complex and is expected to appear in a future paper.

We present a simple approximate derivation of the model, under the assumption that the congestion signal losses are periodic. This arrives at the same mathematical form as the full derivation, although the constant of proportionality is slightly different. This paper is focused on evaluating the model's applicability and impact to the Internet.

The model applies whenever TCP's performance is determined solely by the Congestion Avoidance algorithm (described below). We hypothesize that it applies to nearly all implementations of SACK TCP (TCP with Selective Acknowledgements) [MMFR96] under most normal Internet conditions and to Reno TCP [Jac90, Ste94, Ste97] under more restrictive conditions. To test our hypothesis we examine the performance of the TCP Congestion Avoidance algorithm in three ways. First, we look at several TCP implementations in a simulator, exploring the performance effects of random packet loss, packet loss due to drop-tail queuing, phase effects [FJ92], and Random Early Detection (RED) queuing [FJ93]. Next, we compare the model to Internet measurements using results from the TReno ("tree-no") [Mat96] user mode performance diagnostic. Finally, we compare the model to measurements of packet traces of real TCP implementations.

Many of our experiments are conducted with an updated version of the FACK TCP [MM96a], designed for use with Selective Acknowledgements. We call this Forward Acknowledgments with Rate-Halving (FACK-RH) [MM96b]. Except as noted, the differences between FACK-RH and other TCP implementations do not have significant effects on the results. See Appendix A for more information about FACK-RH.

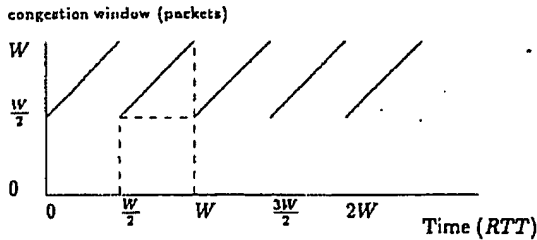


Figure 1: TCP window evolution under periodic loss
Each cycle delivers $(\frac{W}{2})^2 + \frac{1}{2}(\frac{W}{2})^2 = 1/p$ packets and takes $W/2$ round trip times.

2 The Model

The TCP Congestion Avoidance algorithm [Jac88a] drives the steady-state behavior of TCP under conditions of light to moderate packet losses. It calls for increasing the congestion window by a constant amount on each round trip and for decreasing it by a constant multiplicative factor on each congestion signal.¹ Although we assume that congestion is signaled by packet loss, we do not assume that every packet loss is a new congestion signal. For all SACK-based TCPs, multiple losses within one round trip are treated as a single congestion signal. This complicates our measurements of congestion signals.

We can easily estimate TCP's performance by making some gross simplifications. Assume that TCP is running over a lossy path which has a constant round trip time (RTT) because it has sufficient bandwidth and low enough total load that it never sustains any queues. For ease of derivation, we approximate random packet loss at constant probability p by assuming that the link delivers approximately $1/p$ consecutive packets, fol-

lowed by one drop. Under these assumptions the congestion window (cwnd in most implementations) traverses a perfectly periodic sawtooth. Let the maximum value of the window be W packets. Then by the definition of Congestion Avoidance, we know that during equilibrium, the minimum window must be $W/2$ packets. If the receiver is acknowledging every segment, then the window opens by one segment per round trip, so each cycle must be $W/2$ round trips, or $RTT * W/2$ seconds. The total data delivered is the area under the sawtooth, which is $(\frac{W}{2})^2 + \frac{1}{2}(\frac{W}{2})^2 = \frac{3}{8}W^2$ packets per cycle. By assumption, each cycle also delivers $1/p$ packets (neglecting the data transmitted during recovery). Solving for W we get:

$$W = \sqrt{\frac{8}{3p}} \quad (1)$$

Substitute W into the bandwidth equation below:

$$BW = \frac{\text{data per cycle}}{\text{time per cycle}} = \frac{MSS * \frac{3}{8}W^2}{RTT * \frac{W}{2}} = \frac{MSS/p}{RTT \sqrt{\frac{2}{3p}}} \quad (2)$$

Collect the constants in one term, $C = \sqrt{3/2}$, then we arrive at:

$$BW = \frac{MSS C}{RTT \sqrt{p}} \quad (3)$$

Other forms of this derivation have been published [Flo91, LM94] and several people have reported unpublished, "back-of-the-envelope" versions of this calculation [Mat94a, Cla96].

Derivation	ACK Strategy	C
Periodic Loss (derived above)	Every Packet	$1.22 = \sqrt{3/2}$
	Delayed	$0.87 = \sqrt{3/4}$
Random Loss follows [OKM96a]	Every Packet	1.31
	Delayed	0.93

Table 1: Derived values of C under different assumptions.

The constant of proportionality (C) lumps together several terms that are typically constant for a given combination of TCP implementation, ACK strategy (delayed vs non-delayed)², and loss mechanism. Included in the TCP implementation's contribution to C

²The Delayed Acknowledgment ("DA") algorithm [Ste94] suppresses half of the TCP acknowledgments to reduce the number of tiny messages in the Internet. This changes the Congestion Avoidance algorithm because the window increase is driven by the returning acknowledgments. The net effect is that when the TCP receiver sends Delayed Acknowledgments, the sender only opens the window by $MSS/2$ on each round trip. This term can be carried through any of the derivations and always reduces C by $\sqrt{2}$.

The receiver always suppresses Delayed Acknowledgments when it holds partial data. During recovery the receiver acknowledges every incoming segment. The receiver also suppresses Delayed Acknowledgments (or more precisely, transmits acknowledgments on a timer) when the data packets arrive more than 200 ms apart.

There are also a number of TCP implementations which have

are the constants used in the Congestion Avoidance algorithm itself.

The model is not expected to apply under a number of situations where pure Congestion Avoidance does not fully control TCP performance. In general these phenomenon reduce the performance relative to that which is predicted by the model. Some of these situations are:

1. If the data receiver is announcing too small a window, then TCP's performance is likely to be fully controlled by the receiver's window and not at all by the Congestion Avoidance algorithm.
2. Likewise, if the sender does not always have data to send, the model is not likely to apply.
3. The elapsed time consumed by TCP timeouts is not modeled. Many non-SACK TCP implementations suffer from timeouts when they experience multiple packet losses within one round trip time [Flo95, MM96a]. These TCP implementations do not fit the model in environments where they experience such losses.
4. TCP implementations which exhibit go-back-N behaviors do not attain the performance projected by the model because the model does not account for the window consumed by needlessly retransmitting data. Although we have not studied these situations extensively, we believe that Slow-start, either following a timeout or as part of a normal Tahoe recovery, has at least partially go-back-N behavior, particularly when the average window is small.
5. TCP implementations which use other window opening strategies (e.g. TCP Vegas [BOP94, DLY95]) will not fit the model.
6. In some situations, TCP may require multiple cycles of the Congestion Avoidance algorithm to reach steady-state³. As a result, short connections do not fit the model.

Except for Item 6, all of these situations reduce TCP's average throughput. Under many circumstances it will be useful to view Equation 3 as a bound on performance. Given that Delayed Acknowledgements are mandatory, C is normally less than 1. Thus in many practical situations, we can use a simpler bound:

$$BW < \left(\frac{MSS}{RTT} \right) \frac{1}{\sqrt{p}} \quad (4)$$

We will show that it is important that appropriate measurements be used for p and RTT . For example SACK TCP will typically treat multiple packet losses in one RTT as a single congestion signal. For this case, the

bugs in their Delayed Acknowledgment algorithms such that they send acknowledgments less frequently than 1 per 2 data segments. These bugs further reduce C .

³This problem is discussed in Appendix B. All of the simulations in this paper are sufficiently long such that they unambiguously reach equilibrium.

proper definition for p is the number of congestion signals per acknowledged packet.

Although these derivations are for a rather restricted setting, our empirical results suggest that the model is more widely applicable.

3 Simulation

All of our simulations use the LBL simulator, "ns version 1", which can be obtained via FTP [MF95].

Most of the simulations in this paper were conducted using the topology in Figure 2. The simulator associates queuing properties (drop mechanism, queue size, etc.) with links. The nodes (represented by circles) implement TCP, and do not themselves model queues. We were careful to run the simulations for sufficient time to obtain good measures of TCP's average performance⁴.

This single link is far too simple to model the complexity of a real path through the Internet. However, by manipulating the parameters (delay, BW, loss rate) and queuing models (drop-tail, RED) we will explore the properties of the performance model.

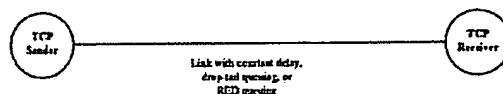


Figure 2: The simulation topologies

3.1 Queueless Random Packet Loss

In our first set of experiments, the single link in Figure 2 was configured to model the conditions under which Equation 3 was derived in [OKM96a]: constant delay and fixed random packet loss. These conditions were represented by a lossy, high bandwidth link⁵ which does not sustain a queue.

The choice of our FACK-RH TCP implementation does not affect the results in this section, except that it is able to remain in Congestion Avoidance at higher loss rates than other TCPs. This phenomenon will be discussed in detail in Section 3.4. The receiver is using standard Delayed Acknowledgements.

The network was simulated for various combinations of delay, MSS , and packet loss. The simulation used three typical values for MSS : 536, 1460, and 4312 bytes. The one-way delay spanned five values from 3 ms to 300 ms; and the probability of packet loss was randomly selected across four orders of magnitude, spanning roughly from 0.00003 to 0.3 (uniformly distributed

⁴This was done by using the bandwidth-delay product to estimate an appropriate duration for the simulation, such that Congestion Avoidance experienced 50 or more cycles. The duration was confirmed from instruments in the simulator.

⁵In order to make sure that the link bandwidth was not a limiting factor, the link bandwidth selected was more than 10 times the estimated bandwidth required. We then confirmed that the link did not sustain a queue.

in $\log(p)$). Since each loss was independent (and assumed to be relatively widely spaced), each loss was considered to be a congestion signal.

In Figure 3 we assume $C = 1$ and plot the simulation vs. the model. Each point represents one combination of RTT , MSS , and p in the simulation. The X axis represents the bandwidth estimated by the model from these measurements, while the Y axis represents the bandwidth as measured by the simulation. Note that the bandwidth, spanning nearly five orders of magnitude, has a fairly strong fit along one edge of the data. However, there are many outlying points where the simulation does not attain the predicted bandwidth.

In Figure 4 we plot a different view of the same data to better illuminate the underlying behaviors. Simulations that experienced timeouts are indicated with open markers. For the remainder of the figures (except where noted), we rescale the Y axis by RTT/MSS . The Y axis is then $BW \cdot RTT/MSS$ which, from classical protocol theory, is a performance-based estimate of the average window size⁶.

We plot p on the X axis, with the loss rate increasing to the right.

To provide a common reference for comparing data between experiments, we plot the line corresponding to the model with $C = 1$ in Figure 4 and subsequent figures.

When $p < 0.01$ (the left side of the graph) the fit between the model and the simulation data is quite plausible. By looking at the data at $p = 0.0001$ we estimate C to be 0.9, which agrees with the Delayed ACK entries in Table 1. Notice that the simulation and model have slightly different slopes, which we will investigate in Section 3.5.

When the average loss rate (p) is large (the right side of the graph), the loss of multiple packets per RTT becomes likely. If too many packets are lost, TCP will lose its Self-clock and be forced to rely on a retransmission timeout, followed by a Slow-start to recover. As mentioned above, timeouts are known not to fit the model. Note that the open markers indicate if there were any timeouts in the simulation for a given data point. Many of the open markers near $p = 0.01$ experienced only a few timeouts, such that the dominant behavior was still Congestion Avoidance, and the model more or less fits. By the time the loss rate gets to $p = 0.1$ the timeout behavior becomes significant. Our choice of FACK-RH TCP alters the transition between these behaviors. We compare different TCP implementations in Section 3.4.

Note that C/\sqrt{p} can be viewed as the model's estimate of window size. This makes sense because packet losses and acknowledgment arrivals drive window adjustments. Although time scale and packet size do determine the total bandwidth, they only indirectly affect the window through congestion signals.

⁶In this paper, "window" always means "window in packets", and not "window in bytes."

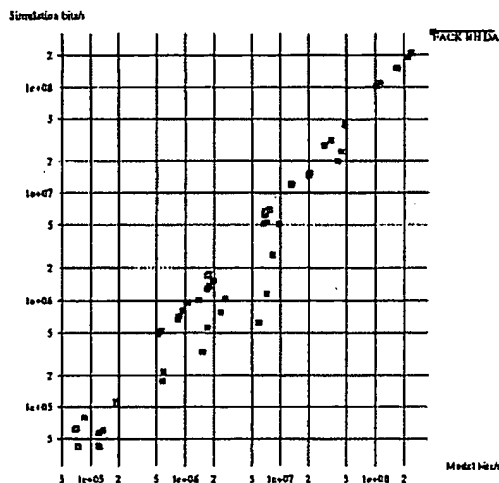


Figure 3: The Measured vs. Estimated BW
The simulation used three typical values for MSS : 536, 1460, and 4312 bytes. The one-way delay spanned from 3 ms to 300 ms; and the probability of packet loss was randomly selected between 0.00003 to 0.3. In the model we have assumed C to be 1.

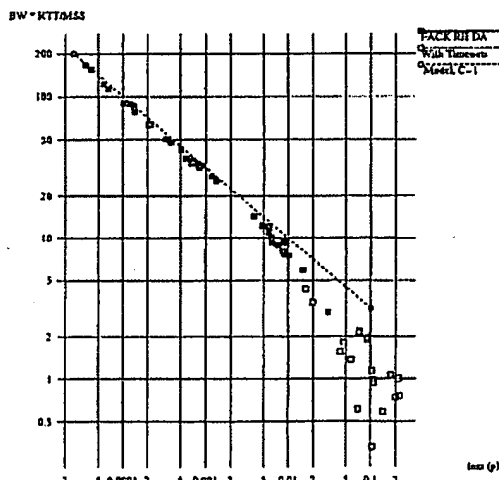


Figure 4: Window vs. Loss
This is a different view of the same data as in Figure 3. Each point has been rescaled in both axes.

3.2 Environments with Queuing

Since, under the assumptions of Section 2, the Congestion Avoidance algorithm is only sensitive to packet loss and Acknowledgement arrivals we expect the model to continue to correctly predict the window when queuing delays are experienced. Thus, with an appropriate definition for RTT , the model should hold for environments with queuing.

We performed a set of simulations using bottlenecked links where queuing could take place. We used a drop-tail link (Figure 2 with drop-tail) with $RTT = 60$ ms and $MSS = 1024$ bytes. The link bandwidth was varied from 10 kb/s to 10 Mb/s, while the queue size was varied from 5 to 30 packets. Therefore, the ratio of delay-bandwidth product to queue length spanned from 15:1 to 1:400. The simulations in Figures 5 and 6 were all performed with the stock Reno module in the simulator.

In Figure 5 we plot the data using the fixed part of the RTT , which includes only propagation delay and copy time. Clearly the fit is poor.

In Figure 6 we re-plot the same data, but use the RTT as measured by a MIB-like instrument in the simulated TCP itself. The instrument uses the round trip time as measured by the RTTM algorithm [JBB92] to compute the round trip time averaged across the entire connection. This is the average RTT as sampled by the connection itself.

This transformation has the effect of making the Y axis a measurement-based estimate of the average window. It moves individual points up (relative to the upper graph) to reflect the queuing delay.

The slope of the data does not quite agree with the model, and there are four clusters of outliers. The slope (which we will investigate in Section 3.5) is the same as in Figure 4.

The four clusters of outliers are due to the long packet times at the bottleneck link causing the Delayed ACK timer to expire. This effectively inhibits the Delayed ACK algorithm such that every data packet causes an ACK, raising C by a factor of $\sqrt{2}$ for the affected points, which lie on a line parallel to the rest of the data.

We conclude that it is necessary to use an RTT measurement that is appropriate for the connection. The RTT as sampled by the connection itself is always appropriate. Under some circumstances it may be possible to use other simpler measures of RTT , such as the time average of the queue at the bottleneck.

Reno fits the model under these conditions because the idealized topology in Figure 2 drops exactly one packet at the onset of congestion⁷, and Reno's Fast

⁷It has been observed that Reno TCP's Self-clock is fragile in the presence of multiple lost packets within one round trip [Hoe95, Flo95, Hoe96, FF96, MM96a, LM94]. In the simulator, a single TCP connection in ongoing Congestion Avoidance nearly always causes the queue at the bottleneck to drop exactly one packet when it fills. This is because the TCP opens the window very gradually, and there is no cross traffic or ACK compression to introduce jitter. Under these conditions Reno avoids any of its problems with closely spaced losses.

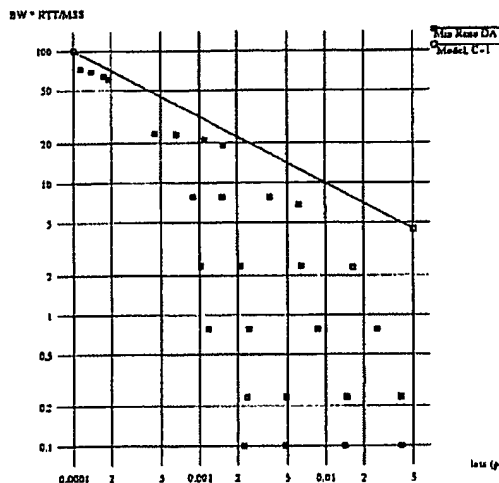


Figure 5: Estimated Window vs. Loss.

Simulations of Reno over a bottlenecked link with a drop-tail queue, without correcting for queuing delay. The RTT was 60 ms and the MSS was 1 kbyte. The bandwidth was varied from 10 kb/s to 10 Mb/s, and the queue size was varied from 5 to 30 packets.

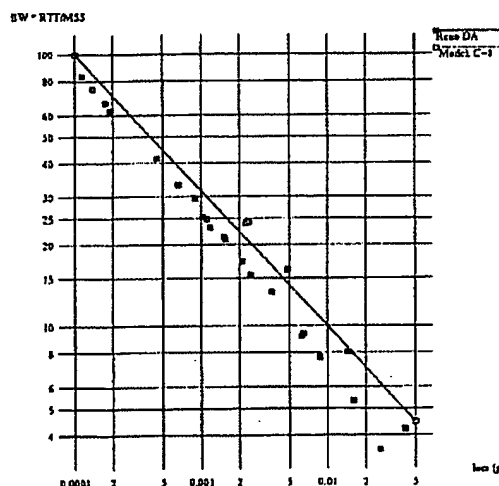


Figure 6: Estimated Window vs. Loss.

This is a different view of the same data as Figure 5, transformed by using TCP's measure of the RTT .

Recovery and Fast Retransmit algorithms are sufficient to preserve the Self-clock. Under these conditions Reno exhibits idealized Congestion Avoidance and fits the model. If the simulations are re-run using other TCP implementations with standard Congestion Avoidance algorithms⁸ the resulting data is nearly identical to Figures 5 and 6. For NewReno, SACK and FACK the data points agree within the quantization errors present in the simulation instruments. This is expected, because all are either derived from the original Reno code, or were expressly designed to have the same overall behavior as Reno when subjected to isolated losses.

3.3 Phase Effects

Phase effects [FJ92] are phenomena in which a small change in path delay (on the order of a few packet times) has a profound effect on the observed TCP performance. It arises because packets leaving the bottlenecked link induce correlation between the packet arrival and the freeing of queue space at the same bottleneck. In this section we will show why phase effects are consistent with the model, and what this implies about the future of Internet performance instrumentation.

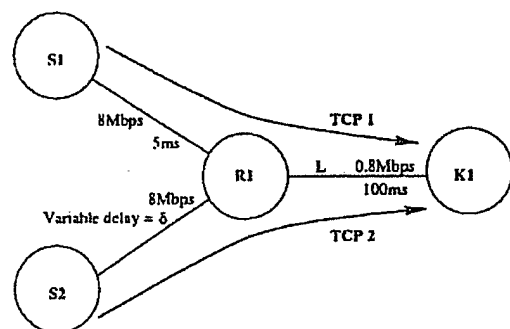


Figure 7: Phase Effects topology.

⁸ The simulator includes models for several different TCP implementations. Tahoe [Jac88a] and Reno (described in [Ste97] and [Jac90]) are well known. The simulator also includes a SACK implementation "SACK1" [Flo98], which was based on the original [JB88] SACK, but has been updated to RFC 2018 [MMFR98]. This is, by design, a fairly straightforward implementation of SACK TCP using Reno-style congestion control. NewReno is a version of Reno that has some modifications to correct what is essentially a bug that frequently causes needless timeouts in response to multiple-packet congestion signals. This modification was first suggested by Janie Hoe [Hoe95, CH95] and has been thoroughly analyzed [Flo95, FF98].

Tahoe TCP has significantly different steady-state behavior than newer TCP implementations. Whenever a loss is detected the congestion window is reduced to 1 (without changing $ssthresh$). This causes a Slow-start, taking roughly $\log_2 W$ round trips, and delivering roughly W segments). Tahoe does not fit the model in a badly underbuffered network (due to persistent repeated timeouts). At higher loss rates when a larger fraction of the overall time is spent in Slow-Start, Tahoe has a slightly different shape, and therefore the model is less accurate.

In Figure 7 we have reconstructed⁹ one of the simulations from [FJ92], using two SACK TCP connections through a single bottlenecked link with a drop-tail router. Rather than reconstructing the complete simulation in which the variable delay is adjusted across many closely spaced values, we present a detailed analysis of one operating point, $\delta = 9.9$ ms. In this case, the packets from connection 2 are most likely to arrive just after queue space has been freed, but just before packets from connection 1. Since the packets from connection 2 have a significant advantage when competing for the last packet slot in the queue, connection 1 sees more packet drops.

The packets are 1 kbyte long, so they arrive at the receiver (node K1) every 10 ms. The 15 packet queue at link L slightly underbuffers the network. We added instrumentation to both the bottlenecked link and the TCP implementations, shown in Table 2. The Link column presents the link instruments on L, including total link bandwidth and the time average of the queue length, expressed as the average queuing delay. The two TCP columns present our MIB-like TCP instruments for the two TCP connections, except for the RTT Estimate row, which is the average RTT computed by adding the average queue length of the link to the minimum RTT of the entire path.

The loss instruments in the TCP implementation categorize each loss depending on how it affected the congestion window. Losses that trigger successful (clock-preserving) divide-by-two window adjustments are counted as "CA events". All other downward window adjustments (i.e. timeouts) are counted as "non-CA events". Additional losses which are detected while already in recovery and do not cause their own window adjustments and are counted as "other losses"¹⁰. In the drop-tail case (on the left side of the table), we can see that TCP1 experienced 103 CA events and 37 non-CA events (timeouts). During those same recovery intervals, there were an additional 76 losses which were not counted as congestion signals. Note that p is the number of CA events per acknowledged segment. The link loss instruments, by contrast, do not categorize lost packets, and cannot distinguish losses triggering congestion avoidance.

The TCP RTT instrument is the same as in the previous section (i.e. based on the RTTM algorithm).

Note that even though the delay is different by only 4.9 ms there is about a factor of 4 difference in the performance. This is because the loss rate experienced by

⁹ Our simulation is identical, except that we raised the receiver's window such that it does not interfere with the Congestion Avoidance algorithm. This alters the overall behavior somewhat because the dominant connection can potentially capture the full bandwidth of the link.

¹⁰ Every loss episode counts as exactly one CA or non-CA event. Episodes in which there was a Fast Retransmit, but Fast Recovery was unsuccessful at preserving the Self-clock or additional losses caused additional window reductions were counted as non-CA events.

All additional retransmissions (occurring in association with either a timeout or congestion signal) are counted as additional lost packets.

Table 2: Phase effects with queue limit = 15, $\delta = 9.9$

Drop Tail			$\delta = 9.9ms$	RED		
Link	TCP1	TCP2		Link	TCP1	TCP2
781	133	648	Bandwidth kb/s	798	430	368
259	103+37+76	41+0+2	losses (CA+timo+other)	139	64+0+1	73+0+1
48795	8287	40508	packets	49853	26851	23002
0.0053	0.0124	0.0010	p	0.0028	0.0024	0.0032
83.49	325	315	TCP RTT ms	77.47	304	307
			Link Delay ms			
	305	315	RTT Estimate ms		299	309
	288 (118%)	279 (57%)	Link Model kb/s		405 (6%)	393 (7%)
	177 (34%)	638 (2%)	TCP Model kb/s		432 (1%)	370 (1%)

Table 3: Phase effects with queue limit = 100, $\delta = 9.9$

Drop Tail			$\delta = 9.9ms$	RED		
Link	TCP1	TCP2		Link	TCP1	TCP2
800	244	556	Bandwidth kb/s	798	401	397
20	12+0+3	5+0+0	losses (CA+timo+other)	137	68+0+0	69+0+0
50000	15250	34750	packets	49845	25039	24806
0.0004	0.0008	0.0001	p	0.0027	0.0027	0.0028
801.03	1029	1029	TCP RTT ms	77.25	304	307
			Link Delay ms			
	1022	1032	RTT Estimate ms		299	308
	313 (29%)	310 (45%)	Link Model kb/s		409 (3%)	396 (1%)
	222 (10%)	518 (7%)	TCP Model kb/s		404 (1%)	395 (1%)

each connection is different by an order of magnitude.

The model is used to predict performance in two different ways. The first technique, the Link Model, uses only the link instruments, while the second, the TCP Model, uses only the TCP instruments. Clearly applying the model to the aggregate link statistics or to TCP1 statistics (with 37 timeouts) in the drop-tail case can not yield accurate results. The model¹¹ applied to TCP2's internal instruments correctly predicts the bandwidth.

Random Early Detection (RED) [FJ93] is a form of Active Queue Management [B⁺97], which manages the queue length in a router by strategically discarding packets before queues actually fill. Among many gains, this permits the router to randomize the packet losses across all connections, because it can choose to drop packets independent of the instantaneous queue length, and before it is compelled to drop packets by buffer exhaustion.

In the phase effects paper [FJ92], it is observed that if a router uses RED instead of drop-tail queuing, the phase effects disappear. In the right side of Table 2 we present a simulation which is identical to the left side, but using RED at the bottleneck (link *L*). With RED, the link instruments are in nearer agreement with the TCP instruments; so the model gives fairly consistent results when calculated from either link statistics or

¹¹ Since we are most interested in the drop-tail simulation near $p = 0.01$, we estimated a locally-accurate value of $C = 0.8$ by examining the data used in Figure 6 in the previous section. This value of C was used for all the model bandwidths shown in Tables 2 and 3.

TCP instruments¹². The residual differences between the results predicted by the model are due to p not being precisely uniform between the two TCP connections. This may reflect some residual bias in RED, and bears further investigation.

In Table 3 we repeated the simulations from Table 2, but increased the packet queue limit at link *L* to 100. As you would expect, this only slightly changes the RED case. However, there are several interesting changes to the drop tail case.

Average RTT has risen to a full second. Without RED to regulate the queue length, SACK TCP only halves its window when it fills the 100 packet queue. TCP's window is being regulated against a full queue, rather than some other operating point closer to the onset of queued data. Even if both connections experience a packet loss in the same RTT, the queue will not fully drain. We can gauge the queue sizes from the average link delay instrument: 800 ms corresponds to 80 packets. We know that the peak is 100 packets, so the minimum queue is likely to be near 60 packets, or 600 ms! This is not likely to please interactive users.

The tripling of the RTT requires an order of magnitude lower loss to sustain (roughly) constant bandwidth. The model is less accurate, even when applied to the TCP instruments because the loss sample size is too small, causing a large uncertainty in p . Excluding the initial Slow-start, TCP2 only experienced 5 losses

¹² Note that RED also lowered the average link delay, lowered the total packet losses, raised the aggregate throughput. RED usually signals congestion with isolated losses. Therefore Reno might operate as well as SACK TCP in this environment.

during the 500 second measurement interval (i.e. each Congestion Avoidance cycle took 100 seconds!)

The symptoms of overbuffering without RED are: long queuing delays and very long convergence time for the congestion control algorithm.

Also note that our opening problem of projecting end-to-end TCP performance from hop-by-hop path properties requires reasonable assurance that the link statistics collected at any one hop are indicative of that hop's contribution to the end-to-end path statistics. This requirement is not met with drop-tail routers, where correlation in the traffic causes correlation in the drops.

If packet losses are not randomized at each bottleneck, then hop-by-hop performance metrics may not have any bearing upon end-to-end performance. RED (or possibly some other form of Active Queue Management) is required for estimating end-to-end performance from link statistics. Conversely, if a provider wishes to assure end-to-end path performance, then all routers (and other potential bottlenecks) must randomize their losses across all connections common to a given queue or bottleneck.

Also note that if the losses are randomized, C/\sqrt{p} is a bound on the window size for all connections through any bottleneck or sequence of bottlenecks. Furthermore, connections which share the same (randomized loss) bottleneck tend to equalize their windows [CJ89]. We suspect that this is the implicit resource allocation principle already in effect in the Internet today.¹³

3.4 Effect of TCP Implementation

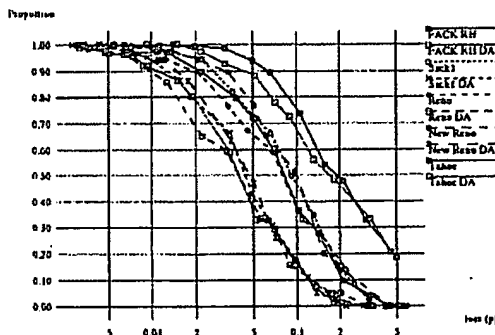


Figure 8: Algorithm dominance vs packet loss. The fraction of all downward window adjustments which are successful (clock-preserving) divide-by-two window adjustments.

¹³Note that our observation is independent of the model in this paper. To the extent that the window is only determined by losses (which isn't quite true) and that losses are equalized at bottlenecks (which also isn't true without RED, etc.), the Internet must tend to equalize windows.

We wish to compare how well different TCP implementations fit the model by investigating two aspects of their behavior. We first investigate the transition from Congestion Avoidance behavior at moderate p to timeout driven behavior at larger p . In the next section, we investigate a least squares fit to the model itself.

As mentioned earlier, the model does not predict the performance when TCP is timeout driven. Although in our simulations timeouts do not cause a serious performance penalty, we have not included cross traffic or other effects that might raise the variance of the RTT and thus raise the calculated retransmission timeout. Although Figure 4 might seem to imply that the model fits timeouts, remember that this was in a queueless environment, where there is zero variance in the RTT . Under more realistic conditions it is likely that timeouts will significantly reduce the performance relative to the model's prediction.

We simulated all of the TCP implementations supported by the simulator⁸, with and without Delayed ACK receivers, and instrumented the simulator to tabulate all downward window adjustments into the same two categories as used in the previous section. The first, "CA events," includes all successful (clock preserving) divide-by-two window adjustments. The second, "non-CA events", includes all other downward window adjustments. In Figure 8 we plot the proportion of all downward adjustments which were successful invocations of the Congestion Avoidance algorithm. (This data is also summarized on the right side of Table 4).

FACK-RH TCP avoids timeouts under more severe loss than the other TCP implementations because it normally sends one segment of new data after the first duplicate acknowledgment but before reaching the dupack threshold (triggering the first retransmission—see Appendix A). All of the other TCP's are unable to recover from a single loss unless the window is at least 5 packets¹⁴. The horizontal position of the steep downward transition reflects the loss rate at which the various TCPs no longer retain sufficient average window for Fast Retransmit. Under random loss SACK, Reno, NewReno, and Tahoe all have essentially the same characteristics.

3.5 Fitting the slope

As we have observed in the previous sections, the window vs. loss data falls on a fairly straight line on a log-log plot, but the slope is not quite $-1/2$. This suggests that a better model might be in the following form:

$$BW = \frac{MSS}{RTT} Cp^k \quad (5)$$

Where k is roughly $-1/2$.

We performed a least mean squared fit between Equation 5 and the TCP performance as measured in the simulator. The results are shown in Table 4. All

¹⁴One packet is lost, the next three cause duplicate acknowledgements, which are only counted. The lost packet is not retransmitted until the fifth packet is acknowledged.

Acknowledgement Scheme	TCP Implementation	Least Mean Squares fit				Proportion of successful W/2 adjustments		
		N	Equation 3 C	Equation 5		p = .01	p = 0.033	p = 0.1
				k	C			
No Delayed ACKs	FACK	16	1.352 ± 0.090	-0.513	1.205 ± 0.047	0.996	0.985	0.738
	SACK	11	1.346 ± 0.052	-0.508	1.247 ± 0.033	0.992	0.822	0.497
	Reno	12	1.331 ± 0.054	-0.521	1.096 ± 0.009	0.935	0.765	0.331
	New Reno	12	1.357 ± 0.055	-0.516	1.167 ± 0.020	0.983	0.896	0.517
	Tahoe	11	1.254 ± 0.079	-0.534	0.920 ± 0.015	0.974	0.796	0.367
Delayed ACKs	FACK DA	15	0.928 ± 0.086	-0.519	0.783 ± 0.045	1.000	0.929	0.725
	SACK DA	10	0.938 ± 0.036	-0.518	0.792 ± 0.012	0.952	0.664	0.112
	Reno DA	10	0.939 ± 0.046	-0.524	0.752 ± 0.015	0.919	0.595	0.157
	New Reno DA	11	0.935 ± 0.045	-0.526	0.738 ± 0.006	0.942	0.635	0.176
	Tahoe DA	11	0.883 ± 0.076	-0.542	0.596 ± 0.012	0.919	0.590	0.173

Table 4: Comparison of various TCP implementations.

simulations which experienced timeouts were excluded, so the fit was applied to runs exhibiting only the Congestion Avoidance algorithm¹⁵. The number of such runs are shown in column *N*.

For $k = -0.5$, the values of *C* are quite close to the derived values. The quality of the fit is also quite good. As expected, Delayed Acknowledgements change *C* by $\sqrt{2}$. When *k* is allowed to vary slightly, the fit becomes even better still, and the best values for *k* are only slightly off from -0.5 . This slight correction to *k* probably reflects some of the simplifying assumptions used in the derivation of Equation 3. One simplification is that TCP implementations perform rounding down to integral values in several calculations which update *cwnd*. The derivation of the model assumes *cwnd* varies smoothly, which overestimates the total amount of data transferred in a congestion avoidance cycle. Another simplification is that the model expects the window to begin increasing again immediately after it is cut in half. However, recovery takes a full *RTT*, during which TCP may not open the window. We plan to investigate the effects of these simplifications in the future.

4 TReno results

Much of our experimentation in TCP congestion dynamics has been done using the TReno performance diagnostic [Mat96]. It was developed as part of our research into Internet performance measurement under the IETF IP Performance Metrics working group [Mat97]. TReno is a natural succession to the windowed ping diagnostic [Mat94b]. (The FACK-RH algorithm for TCP is the result of the evolution of the congestion control implemented in TReno.)

TReno is designed to measure the single stream bulk transfer capacity over an Internet path by implementing TCP Congestion Avoidance in a user mode diagnostic tool. It is an amalgam of two existing algorithms: traceroute [Jac88b] and an idealized version of TCP congestion control. TReno probes the network with ei-

¹⁵FACK fits less well because it avoids timeouts and thus includes data at larger *p*, where rounding terms become significant.

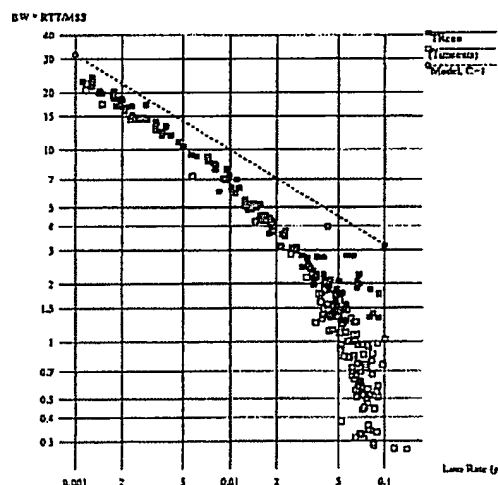


Figure 9: TReno measurement data
This data fits Equation 3 with $C = 0.744 \pm 0.193$ or to Equation 5 with $k = -0.617, C = 0.375 \pm 0.054$. These are poorer than the fits in Table 4, in part because the TReno data extends to much worse loss rates, where the effects of rounding become more pronounced. The *C* values are lower, indicating about 20% lower performance than TCP in a simulator.

ther ICMP ECHO packets (as in the ping program), or low-TTL UDP packets, which solicit ICMP errors (as in the traceroute program). The probe packets are subject to queuing, delay and congestion-related loss comparable to TCP data and acknowledgment packets. The packets carry sequence numbers which are reflected in the replies, such that TReno can always determine which probe packet caused each response, and can use this information to emulate TCP.

This has an advantage over true TCP for experimenting with congestion control algorithms because TReno only implements those algorithms and does not need to implement the rest of the TCP protocol, such as the three-way SYN handshake or reliable data delivery. Furthermore, TReno is far better instrumented than any of today's TCP implementations. Thus it is a good vehicle to test congestion control algorithms over real Internet paths, which are often not well-represented by the idealized queuing models used in simulations.

However, TReno has some intrinsic differences from real TCP. For one thing, TReno does not keep any state (corresponding to the TCP receiver's state) at the far end of the path. Both the sender's and receiver's behaviors are emulated at the near end of the path. Thus it has no way to distinguish between properties (such as losses or delay) of the forward and reverse paths¹⁶.

For our investigation, TReno was run at random times over the course of a week to two hosts utilizing different Internet providers. Due to normal fluctuation in Internet load we observed nearly two orders of magnitude fluctuations in loss rates. Each test lasted 60 seconds and measured model parameters p and RTT from MIB-like instruments.

The TReno data¹⁷ is very similar to the simulator data in Figure 4, except that the timeouts have a more profound negative impact on performance. If the runs containing timeouts are neglected, the data is quite similar.

Also note that TReno suffered far more timeouts than FACK-RH in the simulator, even though they have nearly identical internal algorithms. This is discussed in the next section, where we make similar observations about the TCP data.

5 TCP measurements

In this section, we measured actual TCP transfers to two Internet sites in order to test the model. This was done by using a slightly modified version of "tcptrace" [Ost96] to post-process packet traces to reconstruct p and RTT from TCP's perspective. These instruments are nearly identical to the instruments used in the TCP simulations.

¹⁶ If the ICMP replies originate from a router, (such as intermediate traceroute hops) TReno may suffer from a low performance ICMP implementation on the router. This is not an issue in the data presented here, because the end systems can send ICMP replies at full rate.

¹⁷ TReno emulates Delayed Acknowledgements.

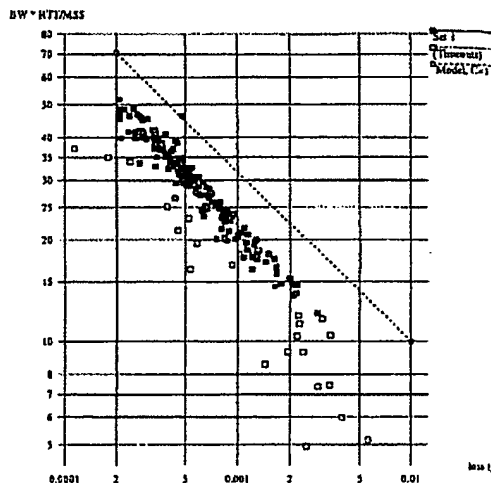


Figure 10: Measured TCP data, Set 1
This data fits Equation 3 with $C = 0.700 \pm 0.057$ or to Equation 5 with $k = -0.525, C = 0.574 \pm 0.045$. These values for C are about 25% lower than TCP in a simulator.

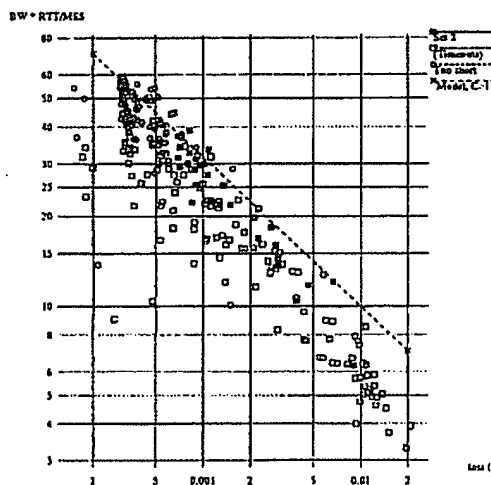


Figure 11: Measured TCP data, Set 2
This data fits Equation 3 with $C = 0.908 \pm 0.134$ or to Equation 5 with $k = -0.611, C = 0.418 \pm 0.058$. Some of the individual data points are above the $C = 1$ reference line. The live Internet measurements were not over long enough intervals to permit trimming the Slow-start overshoot described in Appendix B. This data set may have been subject to Slow-start overshoot.

This experiment proved to be far more difficult than expected. We encountered a number of difficulties with the test paths themselves. The Internet is vastly noisier and less uniform than any simulation [Pax97a, Pax97b]. Furthermore, several paths exhibited behaviors that are beyond the scope of the model¹⁸.

The tests were run at random times over the course of 10 days, at an average rate of once per hour to each remote site. The period included a holiday weekend (with unusually low background traffic), and was not the same week as the TReno data.

During our testing, the connections transferred as much data as possible in 100 seconds of elapsed time to two different Internet sites. Figure 10 shows that the model fits fairly well to data collected to one target. If you compare this to Figure 4 it is in reasonable agreement, considering the difference in scale.

Figure 11, on the other hand, does not fit as well. It is illustrative to dissect the data to understand what is happening over this path, and how it relates to the model's applicability. Our first observation is that there are too many timeouts (indicated by open circles), considering the low overall loss rate.

To diagnose this phenomenon, we looked at the raw packet traces from several of the transfers. Nearly all of the timeouts were the result of losing an entire window of consecutive packets. These short "outages" were not preceded by any unusual fluctuation in delay. Furthermore, the following (Tahoe-style) recovery exhibited no SACK blocks or step advances in the acknowledgment number. Therefore an entire window of data had been lost on the forward path. This phenomenon has been observed over many paths in the Internet [Pax97b, p305] and is present in Figures 9 and 10, as well. Lore in the provider community attributes this phenomenon to an interaction between routing cache updates and the packet forwarding microcode in some commercial routers¹⁹.

Our second observation (regarding traces without timeouts) is that the number of "CA events" is very small, with many traces showing 3 or fewer successful window halving episodes. An investigation of the trace statistics reveals that the path had a huge maximum round trip time (1800 ms), and that during some of our test transfers the average round trip time was as large as a full second. This suggests that the path is overbuffered and there is no active queue management in effect to regulate the queue length at the bottleneck.

Real TCP over this path exhibits the same symptoms as the simulation of an overbuffered link without RED in Section 3.3: long queuing delays and very long cycle times for the congestion avoidance algorithm. As a consequence, our 100 second measurement interval was not really long enough and captured only a few congestion

signals, resulting in a large uncertainty in p . The open circles on the left side of Figure 11 have observable vertical banding in the data corresponding to 1, 2 or 3 total congestion avoidance cycles²⁰. Traces with 4 or more congestion avoidance cycles are included in the good data (solid squares).

Our test script also used conventional diagnostic tools to measure background path properties bracketing the TCP tests. Although we measured several parameters, the *RTT* statistics were particularly interesting. For "not under test" conditions, the minimum *RTT* was 72.9 ms, and the average *RTT* was 82 ms²¹. From the *teptrace* statistics, we know that during the test transfers the average *RTT* rose to 461 ms²².

Our TCP transfers were sufficient to substantially alter the delay statistics of the path. We believe this to be an intrinsic property of Congestion Avoidance: any long-running TCP connection which remains in Congestion Avoidance raises the link delay and/or loss rate to find its share of the bottleneck bandwidth. Then Equation 3 will agree with the actual bandwidth for the connection, and if the losses at the bottleneck are sufficiently randomized, the link statistics (delay and loss) will be common to all traffic sharing the bottleneck.

In general, the current Internet does not seem to exhibit this property. We suspect that this is due to a combination of effects, including Reno's inability to sustain TCP's Self-clock in the presence of closely-spaced losses, the prevalence of drop-tail queues and faulty router implementations.

6 Multiple Congested Gateways

In this section we apply the model to the problem of TCP fairness in networks with multiple congested gateways. Floyd published a simulation and heuristic analysis of this problem in 1991 [Flo91]. In this paper, she analyzed the following problem: given a network of $2n$ gateways, where n are congested by connections that use only one of the congested gateways, what portion of the available bandwidth will a connection passing through all n congested gateways receive? The analysis of this problem presented by Floyd computes bandwidth by determining the packet loss rate for each connection²³. Here, we demonstrate that the same re-

²⁰The bands are at roughly $p = 0.00015, 0.00035$ and 0.0005 .

²¹Each background measurement consisted of 200 *RTT* samples taken either shortly before or shortly after each test TCP transfer. The median of the measurement averages was 80 ms for the "not under test" case.

²²The median of the measurement averages was 468 ms for the "under test" cases.

Unfortunately, the burst losses obscured our background loss rate measurement, because in the average they caused much more packet loss than the true congestion signals. Since they caused TCP timeouts, they were implicitly excluded from the TCP data, but not from our background measurements.

Note that to some extent the burst losses and the RED-less overbuffering are complementary bugs because each at least partially mitigates the effects of the other.

²³We should note that Floyd's work was published three years prior to Mathis [Mat94a] and five years prior to Ott [OKM96a].

¹⁸One discarded path suffered from packet reordering which was severe enough where the majority of the retransmissions were spurious.

¹⁹Note that burst losses and massive reordering are not detectable using tools with low sampling rates. These sorts of problem can most easily be diagnosed in the production Internet with tools that operate at normal TCP transfer rates.

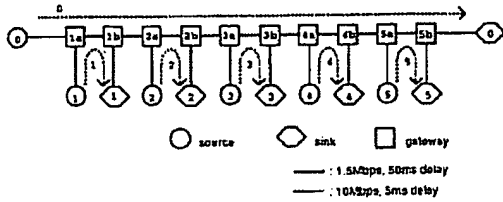


Figure 12: The multiple congested gateways scenario.

sults can be obtained by using Equation 3.

Figure 12 (from [Flo91]) shows the exact scenario we are interested in analyzing. Each dotted line indicates a connection from a source to a sink. Generalizing the parameters, we define δ to be the individual delay of the long links (50 ms in Figure 12) and ϵ to be the delay of the short links (5 ms in Figure 12). The round-trip delay for Connection i is:

$$\delta_i = 2\delta + \delta_Q + 4\epsilon \quad (6)$$

Here, we add a term δ_Q which represents the average delay due to queuing at the bottleneck router.²⁴ The round-trip delay for Connection 0 is:

$$\delta_0 = 2(2n-1)\delta + n\delta_Q + 4\epsilon \quad (7)$$

The model can then be used to predict the bandwidth for each connection:

$$B_i = C \frac{MSS}{\delta_i} \frac{1}{\sqrt{p}} \quad (8)$$

$$B_0 = C \frac{MSS}{\delta_0} \frac{1}{\sqrt{np}} \quad (9)$$

The total link bandwidth used at each congested hop is given by the sum of these two values:

$$B = B_0 + B_i = C \frac{MSS}{\sqrt{p}} \left(\frac{1}{\sqrt{n}\delta_0} + \frac{1}{\delta_i} \right) \quad (10)$$

The fraction of the bandwidth used by Connection 0 is then given by (divide Equation 9 by Equation 10):

$$\frac{B_0}{B} = \frac{1}{1 + \frac{\delta_0 \sqrt{n}}{\delta_i}} = \frac{1}{1 + \sqrt{n} \left(\frac{2(2n-1)\delta + n\delta_Q + 4\epsilon}{2\delta + \delta_Q + 4\epsilon} \right)} \quad (11)$$

The first part of Equation 11 exactly matches Floyd's result [Flo91, Claim 5, Equation 3]²⁵. The second part of Equation 11 fills in the precise formulae for the delays. If we assume δ_Q and ϵ are small, we again match Floyd's results [Flo91, Corollary 6].

²⁴We make the assumption that δ_Q is the same for all of the connections being considered. This is probably not a realistic assumption, but it does allow us to simplify the problem somewhat.

²⁵Noting that we are using an increase-by-1 window increase algorithm, which is identical for both the long and short connections.

$$\frac{B_0}{B} \approx \frac{1}{1 + \sqrt{n}(2n-1)} \quad (12)$$

In the case of overbuffered, drop-tail gateways, where δ_Q is large and phase effects are not an issue, we get a slightly different result:

$$\frac{B_0}{B} \approx \frac{1}{1 + n^{3/2}} \quad (13)$$

It is useful to note that C has dropped out of the calculation. A precise estimate of C was not needed.

In this section, we have used Equation 3 to estimate TCP performance and bandwidth allocation in a complex network topology. Our calculation agrees with the prior heuristic estimates for this environment.

7 Implications for the Internet

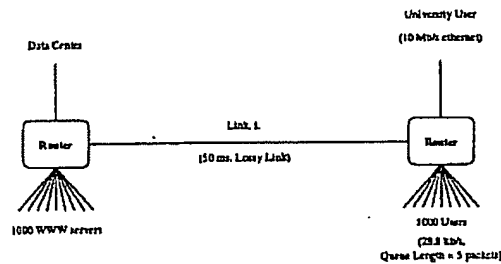


Figure 13: Simplified Internet Topology

In this section we use Equation 3 to investigate some properties of traffic and congestion in the global Internet. Figure 13 is a very simplified schematic of the Internet. To the left are information providers, including a large population of WWW servers and other large data centers, such as supercomputing resources.

To the right are information consumers, including a large population of modem-based users and a smaller population of Research and Education users, who are trying to retrieve data from the data center.

The link between the data suppliers and consumers represents a long path through an intra-continental Internet. For illustration purposes we assume that the path can be modeled as having a single bottleneck, such that the entire path has a fixed average delay and a variable loss rate due to the total load at the bottleneck. Furthermore we assume that each individual data supplier or consumer shown in Figure 13 presents an insignificant fraction of the total load on the link. The loss rate on L is determined by the aggregate load (of many modems and R&E users) on the link and is uniform across all users. Thus the individual connections have no detectable effect upon the loss rate.

We wish to investigate how changes in the loss rate at link L might affect the various information consumers and to explore some strategies that might be used to

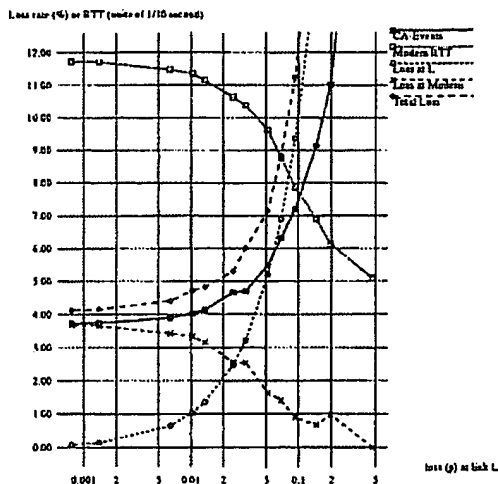


Figure 14: Packet loss experienced by modem users. Packet loss on link L is largely offset by reduced loss at the modem. "Total loss" is computed from the link instruments, "CA-Events" from the instruments in the TCP. (This is for FACK-RH TCP).

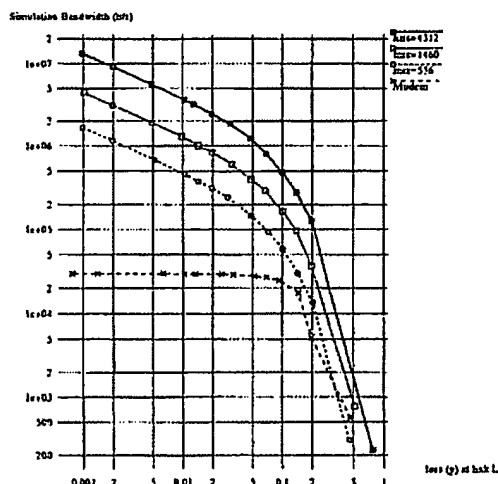


Figure 15: Effect of loss rate at link L . While modem users (with a 1000 byte MSS) do not notice loss on link L until it is quite high, R&E users suffer severe performance degradation, particularly when using a small MSS.

control the elapsed time needed to move scientific data sets.

First we consider TCP congestion control at the modems, which are the likely bottlenecks for the low end users. A full-sized packet (1000 bytes) takes about 277 ms of modem transmission time. The RTT of the unloaded path is about 400 ms (1000 byte packets flowing in one direction, 40 byte acknowledgements flowing in the other). Assume 5 packets of queue space at the modem, then the maximum window size is between 6 and 7 packets, so the "half window" must be roughly 3 packets, yielding an average window of about 5 packets. Since the data packets arrive at the clients with more than 200 ms headway, acknowledgements will be sent for every segment (due to the Delayed Acknowledgement timer). Thus we assume that $C = 1.2$ (the non-Delayed Acknowledgement periodic case from Table 1), so the 5 packet average window requires a loss rate of roughly 4% ($p = 0.04$). This can be observed on the far left edge of Figure 14. (We are assuming FACK-RH TCP).

The packet loss at the modem provides feedback to the TCP sender to regulate the queue at the modem. This queue assures that the modem utilization is high, such that data is delivered to the receiver every 277 ms. These data packets cause the receiver to generate acknowledgements every 277 ms, which clock more data out of the server at a nearly constant average rate.

Next we want to consider what happens if link L loses about 3% of the packets. Since this is not enough to throttle TCP down to 28.8 kb/s, the modem must still be introducing some loss, but less than before. Since the modem is still introducing loss, it must still have a significant average queue, so the server still sends data every 277 ms. With any SACK TCP (including FACK-RH), only the missing data is retransmitted, so the average goodput for the modem user continues to be 28.8 kb/s. Therefore the 3% loss on link L has an insignificant effect on the modem user.

As the loss rate rises beyond 3%, the queue at the modem becomes shorter, reducing the RTT from about 1.2 seconds down toward 400 ms. Note that a 5 packet queue overbuffers the path, so the utilization does not start to fall until the loss rate approaches 10%²⁶.

Now consider the plight of an R&E user (See Figure 15). What performance limitations are imposed on the R&E user by 3% loss? From Equation 3, the average window size must be about 5 packets. If these are 536 byte packets (with a 100 ms RTT), the user can get no more than about 250 kb/s. Timeouts and other difficulties could further reduce this performance. At 250 kb/s, moving 1 Gigabyte of data²⁷ takes over 8 hours.

²⁶Note that this is FACK-RH TCP, which does substantially better than other TCPs in this region.

Many recovery episodes exhibit multiple dropped packets (note that the total link loss rate and CA-Events differ) so Reno has no hope of preserving its Self-clock. As the peak window size falls below 5 packets, conventional Fast Retransmit will also fail.

²⁷Note that at today's prices, with disk space available at below \$100 per Gigabyte, workstations commonly have several Gigabytes of disk space. It is not at all unusual for researchers to

As a consequence, some researchers have been known to express mail tapes instead of using the Internet to transfer data sets.

Suppose the R&E user needs to move 1 Gigabyte of data in 2 hours. This requires a sustained transfer rate of about 1 Mb/s. What loss rate does the user need to meet this requirement? Assume $C < 1$ (because the R&E receivers will be using Delayed Acknowledgments) and invert Equation 4 to get a bound on p :

$$p < \left(\frac{MSS}{BWRTT} \right)^2 \quad (14)$$

The model predicts that the R&E user needs a loss rate better than 0.18% ($p = 0.0018$) with 536 byte packets. At 1460 bytes, the maximum loss rate rises to 1.4%. If the R&E user upgrades to FDDI (and uses 4312 byte packets), Equation 14 suggests that the network only needs to have less than 11% loss.

In practice, we need to consider the actual value of C and potential bottlenecks in all other parts of the system, as well as the details of the particular TCP implementation. This calculation using the model agrees with the simulation shown in Figure 15.

Note that the specific results in this section are very sensitive to many of our assumptions, especially to the use of FACK-RH TCP and the 5 packet queue at the modem. Different assumptions will change the relative positions of the data in our graphs, but the overall trends are due to intrinsic properties of TCP congestion control and the Congestion Avoidance algorithm.

We can draw some useful rules-of-thumb from our observations. First, each factor of 3 in the MSS (4312 to 1460, or 1460 to 536) lowers the required end-to-end loss rate by nearly an order of magnitude. Furthermore, a network which is viewed as excellent by modem users can be totally inadequate for a Research and Education user.

8 Conclusion

We have shown, through simulation and live observations, that the model in Equation 3 can be used to predict the bandwidth of Congestion Avoidance-based TCP implementations under many conditions.

In the simulator all presented TCPs fit the model when losses are infrequent or isolated. However, since different TCPs vary in their susceptibility to timeouts, they diverge from the model at different points.

Live Internet tests show rough agreement with the model in cases where no pathological behaviors are present in the path.

The model is most accurate when using delay and loss instruments in the TCP itself, or when loss is randomized at the bottleneck. With non-randomized losses, such as drop-tail queues, the model may not be able to predict end-to-end performance from aggregate link statistics.

want to transfer a few Gigabytes of data at one time.

FACK-RH, which treats multiple packet losses as single congestion signals, fits the model across a very wide range of conditions. Its behavior is very close to ideal TCP Congestion Avoidance. Reno, on the other hand, stumbles very easily and deviates from the model under fairly ordinary conditions.

To produce a model that applies to *all* loss rates, we need to have a model for timeout-driven behavior.

Overbuffering without RED or some other form of queue management does not interact well with SACK TCP. A single pair of end-systems running SACK over a long Internet path without RED are likely to sustain persistent, unpleasantly long queues.

The model can be used to predict how TCP shares Internet bandwidth. It can also be used to predict the effects of TCP upon the Internet, and represents an equilibrium process between loss, delay and bandwidth.

9 Acknowledgements

We would like to thank Sally Floyd and Steve McCann (LBL's ns simulator), as well as Shawn Ostermann (tcp-trace) for making their software publicly available, without which we would have been unable to complete this work. We would also like to thank Dr. Floyd for allowing us to use Figure 12 from [Flo91]. We appreciate the willingness of Mark Allman, Kevin Lahey, and Hari Balakrishnan to allow us to use equipment at their sites for our remote testing. We would also like to acknowledge Bruce Loftis, for his assistance in fitting parameters to the data, and Susan Blackman, for making suggestions to improve the readability of this paper.

A FACK-RH TCP

The FACK-RH TCP used in the simulations and in the TReno experiment is slightly different than the FACK version presented at Sigcomm96 [MM96a]. We replaced "Overdamping" and "Rampdown" by a combined "Rate-Halving" algorithm, which preserves the best properties of each. Rate-Halving quickly finds the correct window size following packet loss, even under adverse conditions, while maintaining TCP's Self-clock. In addition, we strengthen the retransmission strategy by decoupling it completely from congestion control considerations during recovery. An algorithm we call "Thresholded Retransmission" moves the *tcp_rexmtthresh* logic to the SACK scoreboard and applies it to every loss, not just the first. We also add "Lost Retransmission Detection" to determine when retransmitted segments have been lost in the network.

Rate-Halving congestion control adjusts the window by sending one segment per two ACKs for exactly one round trip during recovery. This sets the new window to exactly one-half of the data which was actually held in the network during the lossy round trip. At the beginning of the lossy round trip *snd.cwnd* segments have been injected into the network. Given that there have been some losses, we expect to receive $(snd.cwnd - loss)$ acknowledgments. Under Rate-Halving we send half as

many segments, so the net effect on the congestion window is:

$$snd.cwnd = \left\lfloor \frac{snd.cwnd - loss}{2} \right\rfloor \quad (15)$$

This algorithm can remain in Congestion Avoidance, without timing out, at higher loss rates than algorithms that wait for half of the packets to drain from the network when the window is halved.

We detect when exactly one round trip has elapsed by comparing the value of the forward-most SACK block in each ACK to the value of *snd.nxt* saved at the time the first SACK block arrived.

Bounding-Parameters add additional controls to guarantee that the final window is appropriate, in spite of potential pathological network or receiver behaviors. For example, a TCP receiver which sends superfluous ACKs could cause Rate-Halving to settle upon an inappropriately large window. Bounding-Parameters assure that this and other pathologies still result in reasonable windows. Since the Bounding-Parameters have no effect under normal operation, they have no effect on the results in this paper.

We are continuing to tinker with some of the details of these algorithms, but mostly in areas that have only minute effects on normal bulk TCP operations. The current state of our TCP work is documented at <http://www.psc.edu/networking/tcp.html>.

B Reaching Equilibrium

In several of our simulations and measurements we noted that an excessive amount of time was sometimes required for TCP to reach equilibrium (steady-state).

One interpretation of Equation 3 is that the average window size in packets will tend to C/\sqrt{p} . However, during a Slow-start (without Delayed ACKs), the expected window size is on the order of $1/p$ when the first packet is dropped, $2/p$ when the loss is detected, and back down to $1/p$ by the end of recovery (assuming SACK TCP). This window is much too large if p is small. It then takes roughly $\log_2 \left(\frac{1/p}{C/\sqrt{p}} \right)$ congestion signals to bring the window down to the proper size. This requires the delivery of $\frac{1}{p} \log_2 \frac{1}{C/\sqrt{p}}$ packets, which is large if p is small.

The effect of this overshoot can be significant. Suppose $p = 1/256$ (approximately 0.004) then we have $1/\sqrt{p} = 16$ and $\log_2 1/\sqrt{p} = 4$. So it takes roughly 1000 packets to come into equilibrium. At 1500 bytes/packet, this is more than 1.5 Mbytes of data.

The average window in steady state will be 16 packets (24 kbytes). If the path has a 100 ms RTT, the steady state average bandwidth will be close to 2 Mb/s. However the peak window and bandwidth might be larger by a factor 16: 256 packet (6 Mbytes) and 30 Mb/s. (This is a factor of $\frac{1/p}{1/\sqrt{p}}$). The overshoot will be this pronounced only if it consumes a negligible fraction of the bottleneck link. Clearly this will not be

the case over most Internet paths so the Slow-start will drive up the loss rate (or run out of receiver window) causing TCP to converge more quickly. It is unclear how significant this overshoot is in the operational Internet.

In all of our simulations we estimate the time required for the connection to reach steady-state, and exclude the initial overshoot when measuring loss, delay and bandwidth.

References

- [B⁺97] Robert Braden et al. Recommendations on Queue Management and Congestion Avoidance in the Internet, March 1997. Internet draft draft-irtf-e2c-queue-mgt-00.txt (Work in progress).
- [BOP94] Lawrence S. Brakmo, Sean W. O'Malley, and Larry L. Peterson. TCP Vegas: New Techniques for Congestion Detection and Avoidance. *Proceedings of ACM SIGCOMM '94*, August 1994.
- [CH95] David D. Clark and Janey C. Hoe. Start-up Dynamics of TCP's Congestion Control and Avoidance Schemes. Technical report, Internet End-to-End Research Group, 1995. Presentation. Cited for acknowledgment purposes only.
- [CJ89] D. Chiu and R. Jain. Analysis of the Increase/Decrease Algorithms for Congestion Avoidance in Computer Networks. *Journal of Computer Networks and ISDN*, 17(1):1-14, June 1989.
- [Cla96] Dave Clark. Private communication, December 1996. Derivation of Bandwidth vs. Loss.
- [DLY95] Peter B. Danzig, Zhen Liu, and Limin Yan. An Evaluation of TCP Vegas by Live Emulation. *ACM SIGMetrics '95*, 1995.
- [FF96] Kevin Fall and Sally Floyd. Simulations-Based Comparisons of Tahoe, Reno and SACK TCP. *Computer Communications Review*, 26(3), July 1996.
- [FJ92] Sally Floyd and Van Jacobson. On Traffic Phase Effects in Packet-Switched Gateways. *Internetworking: Research and Experience*, 3(3):115-156, September 1992.
- [FJ93] Sally Floyd and Van Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, August 1993.
- [Flo91] Sally Floyd. Connections with Multiple Congested Gateways in Packet-Switched

- Networks, Part 1: One-way Traffic. *Computer Communications Review*, 21(5), October 1991.
- [Flo95] Sally Floyd. TCP and Successive Fast Retransmits, February 1995. Obtain via <ftp://ftp.cc.lbl.gov/papers/fastretrans.ps>.
- [Flo96] Sally Floyd. SACK TCP: The sender's congestion control algorithms for the implementation sack1 in LBNL's ns simulator (viewgraphs). Technical report, TCP Large Windows Working Group of the IETF, March 1996. Obtain via <ftp://ftp.cc.lbl.gov/talks/sacks.ps>.
- [Hoe95] Janey C. Hoe. Startup Dynamics of TCP's Congestion Control and Avoidance Schemes. Master's thesis, Massachusetts Institute of Technology, June 1995.
- [Hoe96] Janey C. Hoe. Improving the Start-up Behavior of a Congestion Control Scheme for TCP. *Proceedings of ACM SIGCOMM '96*, August 1996.
- [Jac88a] Van Jacobson. Congestion Avoidance and Control. *Proceedings of ACM SIGCOMM '88*, August 1988.
- [Jac88b] Van Jacobson. Traceroute Source Code, 1988. Obtain via ftp from <ftp://ftp.cc.lbl.gov>.
- [Jac90] Van Jacobson. Modified TCP Congestion Avoidance Algorithm. Email to end2end-interest Mailing List, April 1990. Obtain via <ftp://ftp.cc.lbl.gov/email/vanj.90apr30.txt>.
- [JB88] Van Jacobson and Robert Braden. TCP Extensions for Long-Delay Paths, October 1988. Request for Comments 1072.
- [JBB92] Van Jacobson, Robert Braden, and Dave Borman. TCP Extensions for High Performance, May 1992. Request for Comments 1323.
- [LM94] T.V. Lakshman and U. Madhow. The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss. *IFIP Transactions C-26, High Performance Networking*, pages 135-150, 1994.
- [Mat94a] Matthew Mathis. Private communication, November 1994. Derivation of Bandwidth vs. Loss.
- [Mat94b] Matthew B. Mathis. Windowed Ping: An IP Layer Performance Diagnostic. *Proceedings of INET'94/JENC5*, 2, June 1994.
- [Mat96] Matthew Mathis. Diagnosing Internet Congestion with a Transport Layer Performance Tool. *Proceedings of INET'96*, June 1996.
- [Mat97] Matthew Mathis. Internet Performance and IP Provider Metrics information page. Obtain via <http://www.psc.edu/~mathis/ippm/>, 1997.
- [MF95] Steven McCanne and Sally Floyd. ns-LBL Network Simulator. Obtain via: <http://www-nrg.cc.lbl.gov/ns/>, 1995.
- [MM96a] Matthew Mathis and Jamshid Mahdavi. Forward Acknowledgment: Refining TCP Congestion Control. *Proceedings of ACM SIGCOMM '96*, August 1996.
- [MM96b] Matthew Mathis and Jamshid Mahdavi. TCP Rate-Halving with Bounding Parameters, October 1996. Obtain via: <http://www.psc.edu/networking/papers/FACKnotes/current/>.
- [MMFR96] Matthew Mathis, Jamshid Mahdavi, Sally Floyd, and Allyn Romanow. TCP Selective Acknowledgement Options, October 1996. Request for Comments 2018.
- [OKM96a] Teunis Ott, J.H.B. Kemperman, and Matt Mathis. The Stationary Behavior of Ideal TCP Congestion Avoidance. In progress, August 1996. Obtain via <pub/tjo/TCPwindow.ps> using anonymous ftp to <ftp://ftp.bellcore.com>. See also [OKM96b], August 1996.
- [OKM96b] Teunis J. Ott, J.H.B. Kemperman, and Matt Mathis. Window Size Behavior in TCP/IP with Constant Loss Probability, November 1996.
- [Ost96] Shawn Ostermann. tcptrace TCP dump-file analysis tool. Obtain via <http://jarok.cs.ohiou.edu/software/tcptrace/tcptrace.html>, 1996.
- [Pax97a] Vern Paxson. Automated Packet Trace Analysis of TCP Implementations. *Proceedings of ACM SIGCOMM '97*, August 1997.
- [Pax97b] Vern Paxson. *Measurements and Analysis of End-to-End Internet Dynamics*. PhD thesis, University of California, Berkeley, April 1997.
- [Ste94] W. Richard Stevens. *TCP/IP Illustrated*, volume 1. Addison-Wesley, Reading MA, 1994.
- [Ste97] W. Richard Stevens. TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms, January 1997. Request for Comments 2001.

Connections with Multiple Congested Gateways in Packet-Switched Networks Part 2: Two-way Traffic

Sally Floyd*
Lawrence Berkeley Laboratory
1 Cyclotron Road
Berkeley, CA 94720
floyd@ee.lbl.gov

December 13, 1991

Abstract

In a previous paper [F91] we explored the bias in TCP/IP networks against connections with multiple congested gateways, for networks with one-way traffic. Using simulations and a heuristic analysis, we showed that in a network with the window modification algorithm in 4.3 Tahoe BSD TCP and with Random Drop or Drop Tail gateways, a longer connection with multiple congested gateways can receive unacceptably low throughput. However, we showed that in a network with no bias against connections with longer roundtrip times and with no bias against bursty traffic, a connection with multiple congested gateways can receive an acceptable level of throughput.

In this paper we show that the addition of two-way traffic to our simulations results in compressed ACKs at the gateways, resulting in much more bursty traffic in the network. We use Priority gateways to isolate the effects of compressed ACKs in networks with two-way traffic; these Priority gateways give priority to small packets such as ACK packets, thereby allowing two-way traffic without compressed ACKs. In our simulations with two-way traffic and multiple congested gateways with FIFO service and Drop Tail or Random Drop packet-drop algorithms, the throughput for the connection with multiple congested gateways suffers significantly. This loss of throughput can be reduced either by the use of Priority gateways, which reduces the burstiness of the traffic, or by the use of Random Early Detection (RED) gateways, which are designed to avoid a bias against bursty traffic.

*This work was supported by the Director, Office of Energy Research, Scientific Computing Staff, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

1 Introduction

In this paper we investigate the throughput of connections in TCP/IP networks with two-way traffic and multiple congested gateways.

2 Simulator algorithms

Our simulator is briefly described in [F91], along with the algorithms used in these simulations.

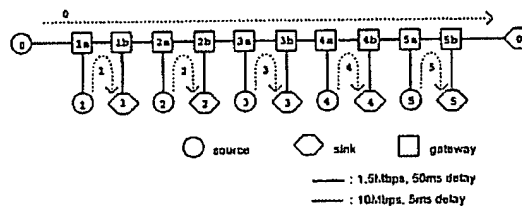


Figure 1: Simulation network with 5 congested gateways.

Figure 1 shows a simulation network for one-way traffic, with 6 connections, 10 gateways, and 5 congested gateways. The congested gateways in Figure 1 are gateways 1a, 2a, 3a, 4a, and 5a. The dotted lines show the connection paths; source i sends to sink i . Each connection has a maximum window just large enough so that, even when that connection is the only active connection, the network still occasionally drops packets.

For one-way traffic we use a family of simulation networks similar to Figure 1, where the number of congested gateways n ranges from 1 to 10. Figure 1 only shows the network for $n = 5$ congested gateways. For

a simulation network with n congested gateways for $n \geq 1$ there are $n + 1$ connections and $2n$ gateways. Connection 0 passes through multiple congested gateways, and connections 1 through n each pass through one congested gateway. Connection 0 is roughly $2n - 1$ times longer than the other n connections. In these simulations all connection paths have the same maximum bandwidth.

Definitions: Reduce-to-One, Increase-by-One. Some of the simulations in the paper use the Reduce-to-One window decrease algorithm and the Increase-by-One window increase algorithm from 4.3 Tahoe BSD TCP [J88]. Our simulator does not use the 4.3-Tahoe TCP code directly but we believe it is functionally identical. Briefly, there are two phases to the window-adjustment algorithm. In the slow-start phase the window is doubled each roundtrip time until it reaches a certain threshold. Reaching the threshold causes a transition to the congestion-avoidance phase where the window is increased by roughly one packet each roundtrip time. In this paper we call the increment algorithm used in the congestion-avoidance phase the Increase-by-One algorithm. Packet loss (a dropped packet) is treated as a "congestion experienced" signal. The source uses timeouts or "fast retransmit" to discover the loss (if four ACK packets acknowledging the same data packet are received, the source decides a packet has been dropped) and reacts by setting the transition threshold to half the current window, decreasing the window to one packet and entering the slow-start phase. In this paper we call this the Reduce-to-One algorithm. \square

In order to achieve the highest throughput for longer connections, some of these simulations use the Fast Recovery algorithm in 4.3 Reno BSD TCP, along with Selective Acknowledgements (or SACKs). In this paper the Fast Recovery algorithm implemented in 4.3 Reno BSD TCP is called the Reduce-by-Half algorithm.

Definitions: Reduce-by-Half window decreases and Selective Acknowledgements. With the Reduce-by-Half window decrease algorithm, when a packet loss is detected by the "fast retransmission" algorithm the connection reduces its window by half. For the purposes of this paper, the important feature of the Reduce-by-Half window decrease algorithm is that with the "fast retransmission" algorithm, the source retransmits a packet and reduces the window by half, rather than reducing its window to one packet. For the simulations in this paper, we use Selective Acknowledgement sinks [JB88] with the Reduce-by-Half algorithm. With Selective Acknowledgement sinks, each ACK acknowledges not only the last sequential packet received for that connection, but also acknowledges all other (non-sequential) packets received. \square

Some of the simulations in this section use the

TCP Increase-by-One window-increase algorithm for the congestion-avoidance phase of the window-increase algorithm. As shown in [FJ91a], this algorithm has a bias against connections with longer roundtrip times. In order to eliminate this bias, some of our simulations use the Constant-Rate algorithm instead in the congestion-avoidance phase. We are *not* proposing the Constant-Rate algorithm for current networks; we simply are using the Constant-Rate algorithm to explore throughput in networks with a window-increase algorithm with no bias in favor of shorter-roundtrip-time connections.

Definitions: Constant-Rate window increases. In the Constant-Rate window-increase algorithm, in the congestion-avoidance phase each connection increases its window by roughly $a * r^2$ packets each roundtrip time, for some fixed constant a , and for r the calculated average roundtrip time. Using this algorithm, each connection increases its window by a pkts/sec in each second. For the simulations in this paper, we use $a = 4$. \square

In this paper we examine networks with Drop Tail, Random Drop, and RED gateways. As shown in [FJ91a], simulations and measurement studies with Drop Tail gateways are vulnerable to traffic phase effects; small changes in network parameters can result in large changes in the performance of the network. In order to avoid these phase effects in networks with Drop Tail gateways, in this paper the simulator adds a small random component to the roundtrip time for each packet in simulations using Drop Tail gateways. (This is discussed in [FJ91a].) Normally, our simulator charges zero seconds for the time required to process packets at the nodes. In this paper each source node adds a random time uniformly distributed in $[0, b]$, for $b \approx 5.3$ ms. the bottleneck service time of the network, to the time required by the source node to process each ACK packet in the simulations with Drop Tail gateways. This is not intended to model any assumptions about realistic network behavior, but to eliminate problems with traffic phase effects. With this added random component the simulations with Drop Tail gateways give similar results to the simulations with Random Drop gateways.

To avoid the bias against bursty traffic common to Random Drop and to Drop Tail gateways, we also examine performance in simulations with Random Early Detection (RED) gateways, a modified version of Random Drop gateways that detect incipient congestion. RED gateways maintain an upper bound on the average queue size at the gateway.

Definitions: RED gateways. With our implementation of RED gateways [FJ91c], the gateway computes the average size for each queue using an exponential weighted moving average. When the average queue size exceeds a certain threshold, indicating incipient

congestion, the gateway randomly chooses a packet to drop and increases the threshold. As more packets arrive at the gateway, the threshold slowly decreases to its previous value. The gateway chooses a packet to drop by choosing a random number n in the interval 1 to $range$, where $range$ is a variable parameter of the gateway. The gateway drops the n th packet to arrive at the gateway. RED gateways are described in more detail in a paper currently in progress [FJ91c]. \square

One advantage of RED gateways is that, unlike Drop Tail and Random Drop gateways, RED gateways do not have a bias against bursty traffic. The bias of Drop Tail and of Random Drop gateways against bursty traffic and the correction of this bias in RED gateways are described in [FJ91a]. With Drop Tail or Random Drop gateways, the more bursty the traffic, the more likely it is that the queue will overflow and the Drop Tail or Random Drop gateway will drop a packet. This is because a burst of packets results in a temporary increase in the queue size at the gateway. With RED gateways the detection of congestion depends on the average queue size, not on the maximum queue size. Thus with RED gateways bursty traffic is less likely to result in the detection of congestion. With a RED gateway even when bursty traffic results in the detection of congestion at the gateway, the mechanism for dropping packets ensures that the bursty connection does not have a disproportionate probability of having a packet dropped. \square

For the RED simulations in this paper the maximum queue size is 60 packets, and the RED gateways drop packets when the average queue size is between 10 and 20 packets. (The range from 10 to 20 packets for the average queue size for RED gateways is somewhat arbitrary; the optimum average queue size is still a question for further research.) For the two-way traffic simulations with Random Drop and Drop Tail gateways, the maximum queue size is 20 packets (I have to rerun these simulations for a maximum queue of 60 packets). The maximum windows for connections 1 to n are set sufficiently large to force occasional packet drops even in the absence of traffic from connection 0.

3 Simulations with two-way traffic

The simulations in Section 2 were of a network with one-way traffic. In this section, we explore simulations with two-way traffic. As discussed in [WRM91] and [ZC91], two-way traffic introduces the added complication of ACK-compression, caused by small ACK packets being queued at a congested gateway with no interleaving larger data packets. This ACK-compression increases the burstiness of the traffic, causing substantial problems for networks with Random Drop (or Drop

Tail) gateways. We show that with Random Drop gateways, the addition of two-way traffic causes a substantial loss in throughput for the longer connection. With RED gateways, with their increased ability to accommodate bursty traffic, the addition of two-way traffic causes only a small change in the behavior of the network. We use gateways with Priority service, which give priority to small packets, to compare the throughput in networks with two-way traffic with and without compressed ACKs.

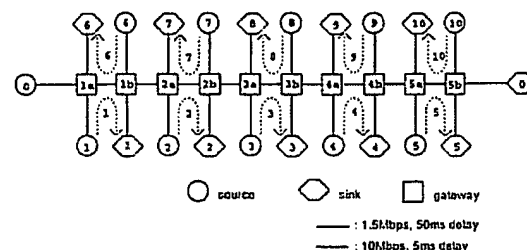


Figure 2: Simulation network with two-way traffic.

Figure 2 shows the simulation network with two-way traffic and with n congested forward gateways, for $n = 5$. Connection 0 goes from source 0 to sink 0. Gateways 1a to 5a are congested in connection 0's forward direction, and gateways 1b to 5b are congested in the reverse direction. With this two-way traffic, connection 0 suffers from significant ACK-compression. All of the connections in this network can experience compressed ACKs, but because connection 0 passes through multiple congested gateways, the problem of compressed ACKs is more severe for connection 0. All of the simulations in this section use Constant-Rate window increases and Reduce-by-Half decreases.

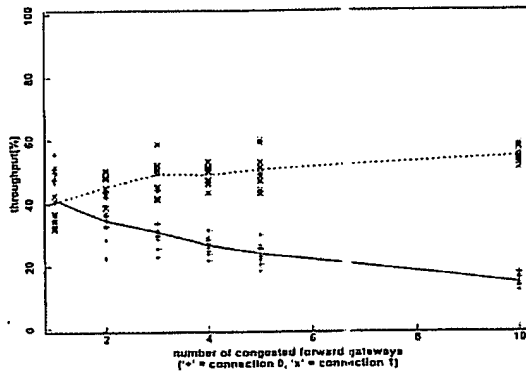


Figure 3: RED gateways, two-way traffic, FIFO service.

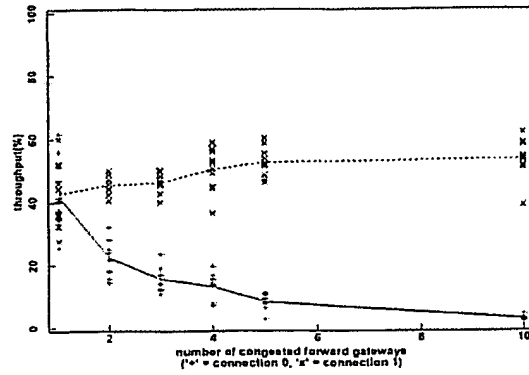


Figure 5: Random-Drop gateways, two-way traffic, FIFO service.

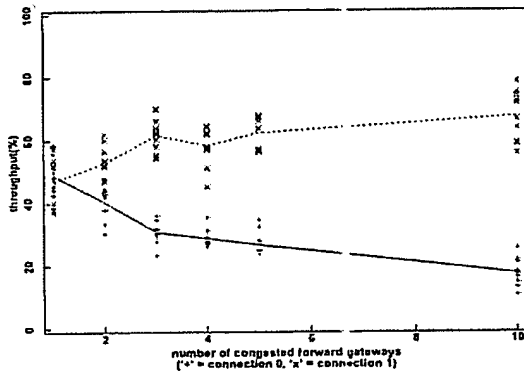


Figure 4: RED gateways, two-way traffic, Priority service.

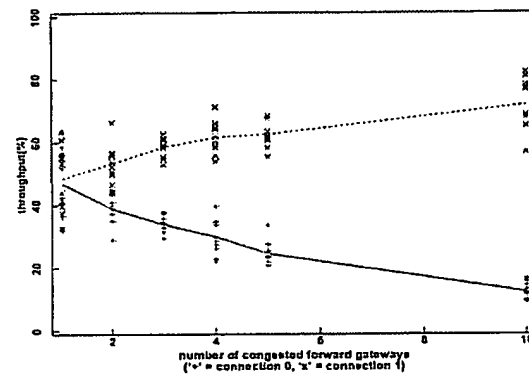


Figure 6: Random-Drop gateways, two-way traffic, Priority service.

Simulations with two-way traffic and RED gateways, with Constant-Rate increases and Reduce-by-Half decreases.

Simulations with two-way traffic and Random-Drop gateways, with Constant-Rate increases and Reduce-by-Half decreases.

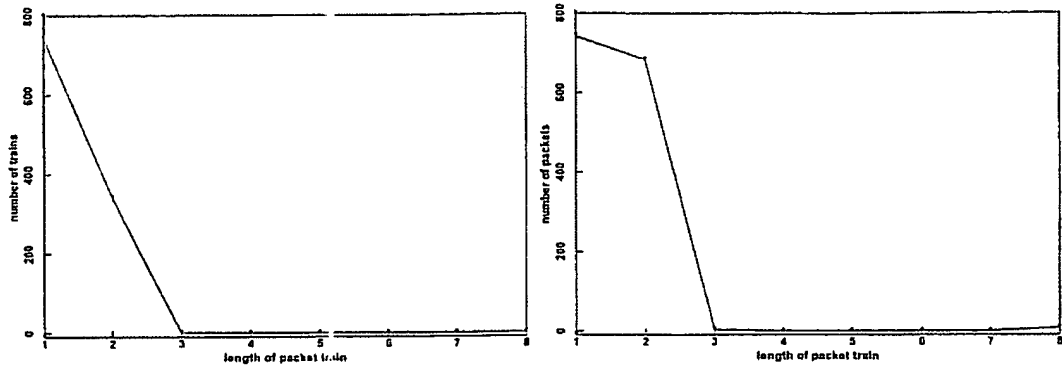


Figure 7: Packet trains with one-way traffic.

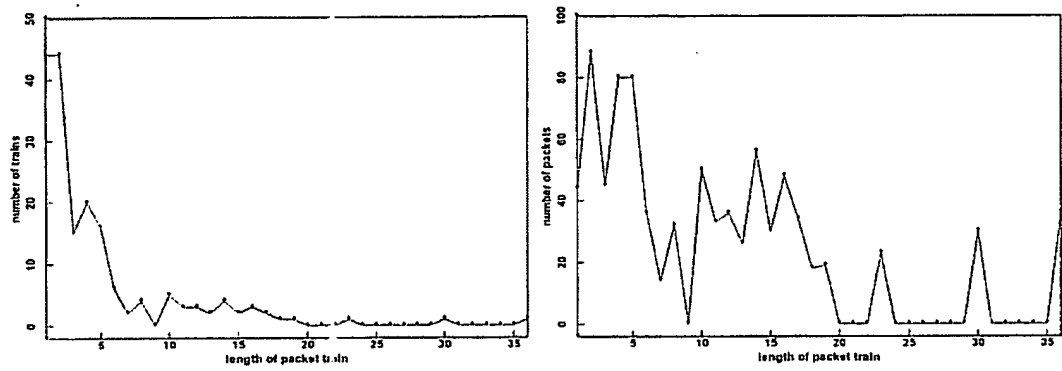


Figure 8: Packet trains with two-way traffic, FIFO service.

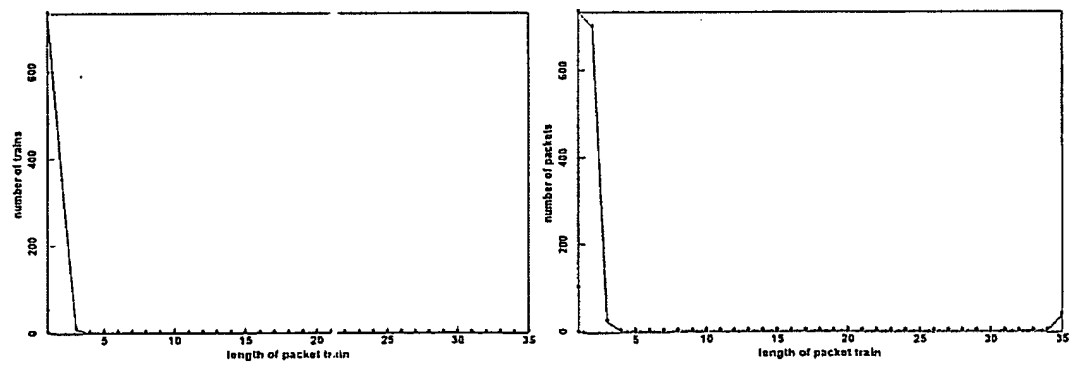


Figure 9: Packet trains with two-way traffic, Priority service.

Figures 3 and 5 show the results of simulations with the network in Figure 2. The simulations in Figure 3 use RED gateways. These differ from the simulations in Figure 6 in Part I only with the addition of traffic in the reverse direction. For the simulations in Figure 3, connections 0 and 1 both receive somewhat reduced throughput, due to the effects of the two-way traffic, but the throughput for connection 0 is still substantial. For the simulations in this section, both the RED and the Random-Drop gateways measure their queues in packets rather than in bytes. Because of this, the addition of two-way traffic adds significantly to the congestion in the network. If our gateways measured queue-length in bytes rather than in packets, this would reduce the impact of two-way traffic.

The simulations in Figure 5 use Random Drop gateways, and differ from the simulations in Figure 6 in Part I only with the addition of traffic in the reverse direction. For the simulations in Figure 5, the throughput for connection 0 is significantly reduced by the addition of two-way traffic, even for a small number of congested gateways. Because of the compressed ACKs for connection 0, the pattern of data packets transmitted by connection 0 is fairly bursty. With Random Drop or Drop Tail gateways, this burstiness increases connection 0's chances of having packets dropped at the gateways. With RED gateways, which are designed to accommodate bursty traffic as much as possible, this burstiness has less of an effect on network behavior.

3.1 Packet trains with two-way traffic

Definitions: packet train. For the network in Figure 1, the bottleneck service time is $b = 5.33$ ms., and it takes a source node $s = 0.8$ ms. to transmit a data packet to the gateway. With one-way traffic, ACK packets arrive at the source node at least b ms. apart, and the source node therefore sends data packets at least b ms. apart, in the absence of window increases. However, the source node is capable of sending data packets at s -ms. intervals. This can occur as a result of a window increase, or as a result of compressed ACKs that arrived at the source node close together. For the purposes of this paper, we define a *packet train* as a sequence of packets that leave the source node at s -ms. intervals, faster than the bottleneck service rate. \square

Figure 7 shows the packet train lengths for connection 0 for the first 50 seconds of a simulation similar to Figure 6 in Part I, with one-way traffic and 10 congested gateways. There are many packet trains of length 1, with single packets, and also many packet trains of length 2, which occur each time connection 0 increases its window by one packet. There is also one packet train of length 3, and one packet train of length 8. These result

from the window opening up after the Reduce-by-Half window decrease algorithm.

With the Reduce-by-Half window decrease algorithm, a connection effectively waits half a roundtrip time after retransmitting a dropped packet, and reduces its window to half of its previous value. If only one packet has been dropped, then the data packets that are transmitted are all clocked by incoming ACK packets. However, due to details of the Reduce-by-Half algorithm, if many packets have been dropped in one roundtrip time, then the Reduce-by-Half window decrease algorithm might open its window to half of its previous value all at once when the ACK packet for the retransmitted packet is received. This results in a burst of packets sent by the source node. Connection 0 in Figure 2 has a large maximum window, and therefore is likely to have many packet drops during the initial slow-start phase of doubling the window each roundtrip time. This accounts for the long packet train in Figure 7 (and Figure 9).

Figure 8 shows the packet train lengths for connection 0 for the first 50 seconds of the simulation as in Figure 3, with two-way traffic and 10 congested forward gateways. There are many packet trains with from 10 to 20 packets, and a few longer packet trains. The lefthand chart of Figure 8 shows the number of packet trains of length m , and the righthand chart shows the number of packets in packet trains of length m . As the righthand chart shows, only a small fraction of the packets are in packet trains of length one or two. Almost all of the packet trains of length greater than two are caused by compressed ACKs.

Two ACK packets are compressed when the first ACK arrives at a gateway to a queue, and the second ACK packet arrives before the first ACK has been transmitted and also before any other intervening packets. Once two ACK packets have been compressed, it is fairly difficult for them to become separated again; another packet has to arrive at some succeeding gateway in the short time between the arrival of the first ACK packet and the arrival of the second ACK packet. Therefore, as the number of gateways with occasional queues increases, the problem of compressed ACKs also increases.

3.2 Priority gateways

One way to demonstrate the effects of compressed ACKs is to run simulations with gateways that give priority to small packets, and to compare these results to those from simulations with gateways with FIFO service.

Definitions: Priority gateways. For gateways with Priority service, we defined *small* packets as being at

most 50 bytes, and we defined *large* packets larger than 50 bytes. In our simulations, all data packets are defined large, and all ACK packets are defined as small. The Priority gateway transmits packets using FIFO service, with one exception. After a Priority gateway sends a large n -byte packet, then if there are any small packets, the Priority gateway sends at most n bytes of small packets before proceeding with the next large packet in the queue. The assumption is that after the current large packet is served, a Priority gateway will generally send all outstanding small packets before sending the next large packet. Thus, small packets that arrive at the gateway queue when the server is busy will usually be transmitted after the gateway finishes transmitting the current large packet. Large packets will have their waiting time in the queue increased by at most a factor of two. Priority gateways are easily implemented, and don't require separate state for each connection. For our simulator, either Priority or FIFO service can be chosen with either Drop Tail, Random Drop, or RED gateways. □

Compressed ACKs are much less likely to occur with Priority service than with FIFO service. With Priority service, each time an ACK packet arrives at a gateway to a busy server, the ACK packet waits at least until the current packet has been served before it can be transmitted. Thus, each Priority gateway adds a certain jitter to the relative timing of the ACK packets. This is quite different from the behavior of FIFO service, where each ACK packet might wait a substantial time in a queue.

Figure 4 shows the results of simulations with RED gateways with Priority service. Connection 0's throughput is somewhat better in Figure 4 than it was in Figure 3. Thus, we can see that the compressed ACKs in Figure 3 did have a (small) effect on throughput in the network. Figure 9 shows the lengths of packet trains for connection 0 for a 50-second simulation as in Figure 4. With Priority service and two-way traffic, almost all packet trains are of length one or two. The one packet train of length 35 is the result of the Reduce-by-Half window decrease algorithm, as explained above.

Figure 6 shows the result of simulations with Random Drop gateways with Priority service. Because the use of Priority service decreases the burstiness of the traffic, the use of Priority service increases the throughput for connection 0 significantly, with Random-Drop gateways. Because the gateways in these simulations use queues measured in packets rather than in bytes, the addition of two-way traffic still significantly increases the congestion in the network, even with Priority service. This accounts for the difference in performance from Figure 6 from Part 1 with one-way traffic, and Figure 6 with two-way traffic with Priority gateways.

The main conclusions of this section are that with

Random Drop (or Drop Tail) gateways, the introduction of two-way traffic can lead to a significant increase in burstiness for traffic with multiple congested gateways, and this increase in burstiness can result in a decrease in throughput. For gateways such as RED gateways that avoid a bias against bursty traffic, the effects of an introduction of two-way traffic are much more moderate. Because Priority gateways allow us to look at networks with two-way traffic but without compressed ACKs, Priority gateways are a useful tool for exploring the effects of compressed ACKs in networks.

We do not necessarily propose implementing Priority gateways in current networks. For a network with Priority gateways, extra care would have to be given to the problems caused by gateways transmitting packets from one connection out-of-order. Packet-switched networks do not guarantee in-order delivery of packets in any case, but the problems of out-of-order packets could be intensified by the use of Priority gateways. This could occur, for example, when some data packets from a connection are small, and other data packets from the same connection are large. (This does not occur in our simulations, because in our simulations all data packets are defined as large, and all ACK packets are defined as small.) At the moment, we are using Priority gateways simply to examine the effects of compressed ACKs at the gateway.

4 Conclusions and Future Work

For our simulations with multiple congested gateways and one-way traffic, the performance of Random Drop gateways and of RED gateways are fairly similar. For our simulations with two-way traffic, however, the traffic for the connection with multiple congested gateways is quite bursty, due to compressed ACKs. In this case, there is a significant difference in performance between Random Drop and RED gateways. Because of their bias against bursty traffic (a bias shared by Drop Tail gateways), the use of Random Drop gateways results in reduced throughput for the connection with multiple congested gateways. For RED gateways, which are designed to accommodate bursty traffic, the introduction of two-way traffic, and the corresponding increase of burstiness, does not result in a significant reduction in throughput for the connection with multiple congested gateways.

In order to explore the effect of compressed ACKs in simulations with two-way traffic, we have introduced gateways with Priority service, which give priority to smaller packets such as ACK packets. With the traditional FIFO service, ACK packets arriving at a congested gateway wait their turn in the queue, and there-

fore it is possible to two successive ACK packets to be "compressed" in the queue. With Priority service, after each data packet is transmitted, (almost all) waiting ACK packets are transmitted before the next data packet in the queue. In this way, with Priority service compressed ACKs are avoided. We show that with Random Drop gateways the throughput for a connection with multiple congested gateways is acceptable in simulations with two-way traffic with Priority service (and no compressed ACKs), but that the throughput is not acceptable in simulations with two-way traffic with FIFO service in the gateway (with compressed ACKs). It is not the two-way traffic itself that degrades performance for the connection with multiple congested gateways, but specifically the compressed ACKs, with the resulting increased burstiness of the traffic.

References

- [BDSY88] Bacon, D., Dupuy, A., Schwartz, J., and Yemini, Y., "Nest: a Network Simulation and Prototyping Tool", *Proceedings of Winter 1988 Usenix Conference*, 1988, pp. 17-78.
- [BJ81] Bharath-Kumar, K., and Jeffrey, J., *A New Approach to Performance-Oriented Flow Control*, IEEE Transactions on Communications, Vol.COM-29 N.4, April 1981.
- [CDJM91] Cáceres, R., Danzig, P., Jamin, S., and Mitzel, D., "Characteristics of Wide-Area TCP/IP Conversations", to appear in SIGCOMM '91.
- [DKS90] Demers, A., Keshav, S., and Shenker, S., "Analysis and Simulation of a Fair Queueing Algorithm", *Internetworking: Research and Experience*, Vol. 1. 1990, p. 3-26.
- [F91] Floyd, S., *Connections with Multiple Congested Gateways in Packet-Switched Networks Part 1: One-way Traffic*, submitted to Computer Communication Review.
- [FJ91a] Floyd, S., and Jacobson, V., *Traffic Phase Effects in Packet-Switched Gateways*, Computer Communication Review, V.21 N.2, April 1991.
- [FJ91c] Floyd, S., and Jacobson, V., *Random Early Detection gateways for congestion avoidance*, in preparation.
- [HG86] Hahne, E., and Gallager, R., *Round Robin Scheduling for Fair Flow Control in Data Communications Networks*, IEEE International Conference on Communications, June, 1986.
- [J88] Jacobson, V., *Congestion Avoidance and Control*, Proceedings of SIGCOMM '88, August 1988.
- [JB88] Jacobson, V., and Braden, R., "RFC 1072: TCP Extensions for Long-Delay Paths", October 1988.
- [JCH84] Jain, R., Chiu, D.M., and Hawe, W., "A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Systems." DEC TR-301, Littleton, MA: Digital Equipment Corporation.
- [K88] Keshav, S., "REAL: a Network Simulator", *Report 88/472*, Computer Science Department, University of California at Berkeley, Berkeley, California, 1988.
- [M90] Mankin, A., "A Measurement Study of Random Drop and Gateway Interaction", 1990.
- [M90a] Mankin, A., *Random Drop Congestion Control*, Proceedings of SIGCOMM 90, Sept. 1990.
- [MS90] Mitra, D. and Seery, J., *Dynamic Adaptive Windows for High Speed Data Networks: Theory and Simulations*, AT&T Bell Laboratories report, April 1990.
- [RJ90] Ramakrishnan, K.K., and Jain, Raj, "A Binary Feedback Scheme for Congestion Avoidance in Computer Networks", *ACM Transactions on Computer Systems*, V. 8, N. 2, p. 158-181, 1990.
- [R83] Ross, S., *Stochastic Processes*, John Wiley & Sons, 1983.
- [WRM91] Wilder, R., Ramakrishnan, K.K., and Mankin, A., "Dynamics of Congestion Control and Avoidance in Two-Way Traffic in an OSI Testbed", SIGCOMM '91, April 1991, p.43-58.
- [ZC91] Zhang, L., and Clark, D., "Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic", to appear in SIGCOMM '91.

**This Page is Inserted by IFW Indexing and Scanning
Operations and is not part of the Official Record**

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☒ **BLACK BORDERS**
- ☐ **IMAGE CUT OFF AT TOP, BOTTOM OR SIDES**
- ☐ **FADED TEXT OR DRAWING**
- ☐ **BLURRED OR ILLEGIBLE TEXT OR DRAWING**
- ☐ **SKEWED/SLANTED IMAGES**
- ☒ **COLOR OR BLACK AND WHITE PHOTOGRAPHS**
- ☐ **GRAY SCALE DOCUMENTS**
- ☒ **LINES OR MARKS ON ORIGINAL DOCUMENT**
- ☐ **REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY**
- ☐ **OTHER:** _____

IMAGES ARE BEST AVAILABLE COPY.

As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.